IMPROVING RESOURCE UTILIZATION IN A PARTITIONABLE BUS NETWORK USING GRAPH COLORING AND COIN-CHANGING ALGORITHMS

By

TAI-KUO WOO

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1989

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my adviser, Dr. Stanley Y.W. Su, for his advice and guidance. Many thanks are also due to Dr. Yuan-Chieh Chow, Dr. Herman Lam, Dr. Richard Newman-Wolfe, and Dr. Chung-Yee Lee for their encouragement. Without their help, this work would not have been finished.

I am also grateful to David Johns for his proofreading of this dissertation. Lastly, I sincerely thank my roommate, David Kern, who has helped in making this endeavor possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

IMPROVING RESOURCE UTILIZATION IN A PARTITIONABLE BUS NETWORK USING GRAPH COLORING AND COIN-CHANGING ALGORITHMS

By

Tai-Kuo Woo

May 1989

Chairman: Dr. Stanley Y. W. Su
Major Department: Computer and Information Sciences

Achieving efficiency in a parallel processing environment is a fundamental problem in many computer science and engineering disciplines. In a large scale computer system, resources such as memory, secondary devices, and communication networks are usually shared by processors and processes. Resource contentions often occur, and system performance is degraded due to blocking and deadlock problems.

System performance can be improved by detecting non-conflicting processes and scheduling them for parallel processing. In this dissertation, we introduce three graph coloring algorithms for distinguishing conflicting and non-conflicting processes. The complexity of each algorithm is O(E), where E is the number of edges of the graph. By interpreting the results of the graph traversal algorithm, non-conflicting processes can be scheduled for parallel communication or processing.

Another problem dealt with in this work is the idling problem in the execution of non-conflicting processes. Since the processes may take different amount of times to execute, if processors are assigned to process them to their completion, the processors of shorter processes will be idle after the completion of their tasks. In this dissertation,

a coin-changing algorithm is applied to achieve better scheduling of non-conflicting processes.

Both the graph traversal algorithms and coin-changing algorithms are then applied in a dynamically partitionable bus network to demonstrate that non-conflicting communication requests can be identified and scheduled for execution in partitioned subnetworks. The performance of a dynamically partitionable bus network using these algorithms is evaluated.

This work makes the following specific contributions. First, it introduces and analyzes three graph coloring algorithms and their performance. An analytical study shows that the dynamic graph traversal algorithm has better performance than other existing graph coloring algorithms. Second, it presents a design of a hardware for implementing the dynamic graph traversal algorithm to meet the time requirement of some real-time environment. Third, it demonstrates the utility of the graph traversal algorithms in a partitionable bus network by analysis and simulation. Fourth, it provides an analysis of a coin-changing algorithm and its application to solve the bus idling problem in a partitionable bus network.

CHAPTER 1
INTRODUCTION

Parallel processing is an efficient form of information processing. It allows several concurrent processes to be processed simultaneously. System resources such as communication networks, memory devices, and processors are shared among processes. One of the limiting factors to the expansion of parallel processing is performance degradation due to resource contentions that occur when multiple processes request the same resources.

Local area networks are example systems in which resource contentions degrade system performance. They are characterized by a high bandwidth transmission medium shared by many interconnecting processors within a limited area. In the past, local area networks have been developed to provide limited functions such as message passing and file transfer, and the amount of data transmitted among processors is small. If the number of computers attached to the network is large and the volume of communication among processors/memory devices increases, conventional interconnection devices, such as a single shared bus/ring, can not meet the increasing demands. Consequently, bus contention is unavoidable, and system performance degrades rapidly because of long communication delays.

Much effort has been made in developing efficient local area networks, such as the development of protocols to reduce contentions in CSMA/CD (carrier sense multiple access with collision detection) [37] and token passing [23], and the design of hardware in register insertion ring networks [17]. An alternative approach is to partition a communication bus and to allocate communication networks dynamically.

The concept of partitioning a computer system was introduced in [3,26,54]. Kartashev and Kartashev [26] propose varying word size to meet the requirements of the task on hand by physically partitioning registers and memory words. Su and Baru [3,54] use a physically partitionable bus to allow for the formation of a number of clusters of processors for parallel processing of database operations. However, the formation of clusters of processors is based on the data distribution and processing power needed. No specific algorithm is used to resolve the conflicts in the formation of clusters. In other words, the scheduling of database operations is based on a first-come-first-served basis. Thus, the degree of parallelism is not extended to its potential limit.

In this dissertation, three graph coloring algorithms are proposed for identifying conflicting and non-conflicting communication requests. By assigning non-conflicting requests to different subnetworks created by manipulating switches of a dynamically partitionable bus, parallel communication can be achieved and maximized. This dissertation also presents a hardware design for implementing one of the algorithms. The performance evaluation of the partitionable bus network that uses a graph coloring algorithm for subnetwork formation is also presented.

Another problem associated with the dynamically partitionable bus network is bus idling. In a dynamically partitionable bus network, the non-conflicting communication requests being carried out at the same time in the subnetworks usually are of different time lengths. The subnetworks assigned to the requests that finish early have to wait for other communication processes to finish. As a result, the subnetworks are idle and are not fully utilized. The bus idling problem can be eliminated by using a coin-changing algorithm. The strategy works as follows. First, the length of each communication request is partitioned into a number of time frames of different sizes. For instance, if the set of frame sizes is (1, 5, 10, 15), a communication

request of length 70 can be partitioned into 4 frames of size 15 and 1 frame of size 10. Each time when the requests are collected for graph traversals, only the requests that have been assigned a tag frame can join the graph traversals. The tag frame is designated in turn from the largest frame size to the smallest frame size (i.e., $15 \rightarrow 10 \rightarrow 5 \rightarrow 1$). There are many ways a request can be decomposed. For instance, the length 70 can be decomposed into either 2 frames of size 15 and 4 frames of size 10 or 4 frames of size 15 and 1 frame of size 10. The objective is to use the minimum number of frames for a given set of frame sizes, since a large number of frames will result in high frequency of graph traversals. The problem of minimizing the number of time frames can be transformed into a coin-changing problem which minimizes the number of coins for a given set of coin types. A performance evaluation shows a significant increase of network throughput by applying the coin-changing algorithm to the dynamically partitionable bus network.

The approach presented in this dissertation can be outlined as follows:

1. Decompose each communication request into a number of time frames by a coin-changing algorithm.

2. Transform communication requests into a graph where vertices represent communication requests and edges denote the conflicts among communication requests.

3. Use graph coloring techniques to color the constructed graph so that non-conflicting communication requests are assigned the same color.

4. Manipulate the switches of the partitionable bus network to form a number of subnetworks to allow parallel communication among requests that have been assigned the same color.

This dissertation is organized as follows: Chapter 2 gives a survey of related work in local area networks, partitionable bus networks, graph coloring algorithms, and coin-changing algorithms. Chapter 3 presents the graph traversal algorithms and the analyses of algorithms. Chapter 4 delineates the application of the best graph traversal algorithm in a dynamically partitionable bus network (DPBN), the hardware design of graph traversal unit (GTU), and the performance evaluation of the DPBN. Chapter 5 provides a coin-changing analysis and its application to the dynamically partitionable bus network. A conclusion is given in Chapter 6.

CHAPTER 2
SURVEY OF RELATED WORK

This dissertation touches a number of areas namely, local area networks, partitionable bus networks, graph coloring algorithms, and coin-changing algorithms. This section provides a brief survey of these areas.

2.1 Local Area Networks

There are two most common topologies of local area networks: bus and ring. Channel sharing is one of the communication techniques used in both topologies.

In a bus network, only one communication is allowed at a time. CSMA/CD and token bus are good examples of protocols used on this topology. In the token bus architecture, access to a shared bus is achieved by passing a token from station to station. The order in which the token is passed follows a "logical ring," which can be modified dynamically as stations are added to or dropped from the bus, and reestablished when token pass failures occur. A token regulates the right to access the bus. When a station receives the token, it is granted control of the medium for a period of time. When the packet transmission is over, it passes the token to the next station in a predefined sequence. This protocol requires considerable maintenance, which includes network initialization, addition to the network, deletion from the network, and fault management (multiple tokens, missing token). In order to meet the above requirements, the logic of each station can be very complex, and the overhead involved in token passing is a waste of bandwidth. In CSMA/CD, also known as listen before (while) talk, each station listens to the medium to determine if the medium

is available. The CD part indicates that a station listens to its own transmission to detect collisions. If the medium is idle, the station may either transmit immediately or transmit with a probability. If the medium is busy, the station may either continue to listen until the medium is sensed idle, transmit with a probability, or wait a random period before sensing the channel. When collisions occur, the station immediately stops transmitting messages and waits a random period of time before retransmission. A major disadvantage of CSMA/CD is the non-deterministic behavior of the bound of delay. It is possible that a station may never be able to transmit messages when the load of the network is heavy.

In a ring network, for instance, the Cambridge Ring [42], there is a sequence of point-to-point links between stations. Messages travel over a fixed route from station to station around the loop in which each station is responsible for regenerating messages and identifying addresses. In some cases, packets can be sent along different links simultaneously. Ring access protocols play a major role in the performance of ring networks. There are three basic types of ring access protocols: token passing, empty slot, and register insertion. The token ring protocol uses a token that circulates around the ring. Each station wishing to transmit messages must wait for a free token passing by. When the transmission is over, the station passes the token to the physically adjacent station, as opposed to the logically adjacent station as in token bus networks. The disadvantages are similar to token bus networks: only one station is able to transmit messages at a time, and fault management increases the complexity of the network.

In the slotted ring, a number of fixed-length slots circulate around the ring. A station finds an available empty slot and inserts a packet of messages. When the full slot reaches its destination, the leading bit of a full slot is set to empty and is free for transmission again. The main disadvantages of slotted rings are: (1) overhead bits

take a significant amount of space, and (2) a data packet may not occupy the entire slot.

Each station in the register insertion ring has a shift register, a buffer which temporarily holds the packet passing through the station, and a buffer which stores the packets produced by the station. Packets which arrive at the shift register are received if the station is the addressee. For the output from the station, packets are placed in the buffer first and then transmitted to a shift register when it is idle. The fault management requires a significant amount of work.

No matter what protocol is used, the following three major problems still exist in ring networks [19]:

1. The operation of the network depends on each station's network controller to regenerate the message. Thus, if an interface or controller fails, the network is essentially broken. This addresses the concern for reliability. Pierce [44] suggests using a "simple circuit" to bypass failed stations. Liu of Ohio State University [33] proposes a distributed double-loop computer network (DDCLN) as a fault-tolerant distributed system. Both approaches result in a higher hardware cost.

2. A ring network must be broken to add or delete stations. Expansion of the network may result in an interruption of network service.

3. Propagation delay is proportional to the number of stations in the network. As the number of stations increases, propagation delay can be a serious problem.

Register insertion ring networks [17] allow a certain degree of parallelism; however they require a minimum of a 7-bit station latency for addressing. On the other hand, only one bit delay is sufficient for slotted and token rings. Long addressing delay

damages the performance of register insertion networks. All in all, the existing local area networks have defects and need to be improved. A partitionable bus network that allows multiple communication processes to be carried out at the same time is a possible solution.

2.2 Partitionable Bus Networks

As discussed above, both types of networks (bus and ring) have disadvantages. The lack of parallel communication has greatly reduced the performance of existing networks. The idea of bus partitioning was introduced in database machines to speed up data processing. There are two kinds of partitioning: logical partitioning and physical partitioning. First, in logical partitioning as used in DIRECT [11], a backend computer allocates query processors to perform tasks and handles data requests from processors. The system is divided into a number of subsystems logically and a controller monitors the progress of each subsystem. This scheme has two weak points as mentioned in [55]. First, the controller may become the bottleneck, since the control of operations is centralized. Second, the data movement between the mass storage and the main memory of the query processors can cause memory contentions in the interconnection network.

In contrast to logical partitioning, physical partitioning reconfigures the hardware of the computer to maximize system efficiency. Kartashev and Kartashev [26], of Dynamic Computer Architecture, Inc., vary the word size by connecting computer elements to meet the requirements of processing tasks on hand. Each computer element consists of a processor element, a memory element, and an I/O element. Su and Baru, at the University of Florida Database Systems Development and Research Center, use a physically partitionable bus in a system called SM3 [3,54] to increase the degree of parallelism. It has three important features. First, data movement among

memory modules is done by memory switching instead of transmitting data among buses. Second, inter-processor communication and synchronization is achieved by using control lines. Third, concurrent processes are executed in parallel in the clusters of processors formed by physically partitioning a common bus. The formation of clusters of processors is based on the data distribution, processing power needed, and file sizes. No specific algorithm is used to resolve the conflicts in the formation of clusters. Thus, the degree of parallelism is not maximized. This dissertation uses a fast graph coloring algorithm for distinguishing conflicting and non-conflicting communication requests and then adopts the idea of a partitionable bus to form a number of clusters of processors for processing the non-conflicting communication requests simultaneously. Thus, the system throughput can be significantly improved.

2.3 Graph Coloring Algorithms

Increasing system throughput and reducing response time are important objectives in designing and implementing a computer system. By duplicating hardware devices and allowing computation tasks to be processed in these devices in parallel, the degree of parallelism can be increased. However, the cost of such a hardware system can be rather high. An economic alternative for achieving a high degree of parallelism is to distinguish conflicting and non-conflicting events by software through graph coloring techniques and to execute non-conflicting events in parallel on a shared hardware resource.

There are two classes of graph coloring problems: vertex coloring and edge coloring. The vertex coloring of a graph is the assignment of colors to the vertices of a graph so that no two adjacent vertices have the same color. Edge coloring is defined in a similar fashion: two edges sharing a common vertex in a graph G must be assigned with different colors. It has been proven that a graph G can be edge colored

with N colors if and only if the vertices of G can be colored with N colors [41]. Thus, an edge coloring problem can be transformed into a vertex coloring problem. For the rest of this dissertation, the graph coloring problem is defined as a vertex coloring problem.

The vertex coloring technique can be applied in computer systems in the following way. If we represent the vertices in a graph as the events of a computer system and edges as the conflicts among events, the event scheduling problem can be transformed into a vertex coloring problem. By finding the minimum number of colors for the vertices of a graph and scheduling the events (or vertices) of the same color for parallel execution, a higher degree of parallelism in a computer system can be achieved. Finding the minimum number of colors (i.e., the chromatic number) required for a given graph is an NP-complete problem [15,25]. However, it is possible to find algorithms that can assign colors to any arbitrary simple graph (i.e., a graph without self-loops nor multiple edges between any pair of vertices) and the number of colors assigned is close to the chromatic number. We shall survey the existing methods that attempt to find the exact chromatic number and the approximate chromatic number below:

1. Exact chromatic number: Algorithms in this category find the chromatic number of a graph. However, they require a very large amount of computing time and memory space. For example, Christofides' algorithm [9] constructs a maximum subgraph tree for a given graph from its "maximal independent sets." Through a breadth-first search method, the shortest path from the root to the terminal nodes can be found. The maximal independent sets on the shortest path are the color classes of chromatic coloring. Unfortunately, the subtree constructed by this algorithm can be too large to be practical for implementation.

Christofides' algorithm was improved by Wang [59], who showed that, by considering only a subset of all the maximal independent sets of a graph, the required number of steps to compute the chromatic number of a graph can be reduced by a factor of as much as ($| V |/ 2 !$), where $| V |$ is the number of vertices of a graph and 2 ! denotes 2 factorial. By using a depth-first search method, the amount of memory required to find the shortest path from the root to the terminal nodes can be reduced by a ratio of ($| V |/8!$) to ($| V |/2!$). However, the computing time of this algorithm grows exponentially with the increase of the size of the graph.

2. Approximate chromatic number: Due to the intractability of the graph coloring problem, some algorithms produce a larger number of colors for a graph than it is necessary. For instance, Greedy's algorithm orders the vertices in arbitrary order, say, $V_1, V_2, V_3, ..., V_n$, and then colors the vertices in sequence. First, color V_1 with color 1, then check if V_1 and V_2 are non-adjacent. If so, color V_2 with color 1, otherwise with color 2. Continue this process until all the vertices are colored. The result of the Greedy algorithm is highly dependent on the order of vertices.

   Brooks [6] provides a fast way of computing the upper bound for the chromatic number of a graph. The upper bound is defined as

   $$max\ (d_1, d_2, d_3, ..., d_n)$$

   where d_i is the degree of vertex i. The estimation is valid only if the graph G(V , E) is not a complete graph and the largest degree of the vertex of the graph is greater than or equal to three.

The method introduced by Welsh and Powell [60] first arranges the vertices in descending order of their degrees (i.e., d1 $\geq$ d2 $\geq$ d3,...,dn). It then colors the first vertex with color 1 and sequentially colors the rest of vertices which are not adjacent to a previously colored vertex. This process is repeated until all the vertices are colored.

Welsh and Powell's algorithm provides an upper bound for the estimation of the chromatic number of a graph:

$$max_i \; min \; (d_i + 1, i)$$

where i is the index of the vertices arranged in descending order of degrees.

The example below illustrates the procedure for computing the upper bound of the chromatic number of the graph as shown in Figure 2.1.

The following steps constitute the procedure:

Step 1: Arrange the degrees of the vertices in descending order.

$$d_i$$

4 3 3 2 2

Step 2: Select the minimum for each pair of values $(d_i + 1, i)$, where i goes from 1 to 5.

$$min \; (d_i + 1, i)$$

1 2 3 3 3

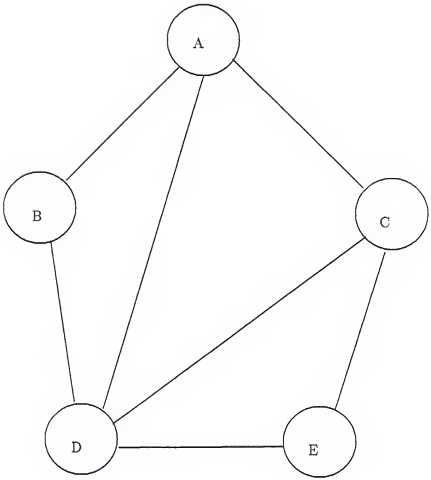


Figure 2.1. An Example of Computing the Upper Bound of a Graph Using Welsh and Powell's Algorithm

Step 3: Choose the largest value among the values in Step 2.

$$max_i\ min\ (d_i + 1, i) \quad = \quad 3$$

In this example, it happens to be the chromatic number of the graph. This algorithm produces good results in some cases. However, the number of colors required may increase if the order of vertices is changed. To correct this problem, Wood [61] proposes a coloring method by forming a similarity matrix. A similarity matrix S = $\{s_{ij}\}$ is formed to determine which vertices should be colored with the same color. The values of s_{ij} are determined by the conflicts of vertices. $s_{ij} = 0$ if $c_{ij} = 1$. $s_{ij} = \sum_k (c_{ik} \cap c_{jk})$ if $c_{ij} = 0$. It should be noted that k is summed over all vertices and C = $\{c_{ij}\}$ is the conflicting matrix, where c_{ij} = 1 if vertex i and vertex j have a conflict, and $c_{ij} = 0$ if vertex i and vertex j have no conflict.

To color the vertices of a graph, assign a color to the pair of vertices with the highest similarity based on whether both, one, or none of the vertices has been colored. Each time the similarity matrix is scanned, the similarity level is reduced by one. Continue this process until all the vertices are colored. The similarity matrix method is better than Welsh and Powell's method when the number of vertices is large and the graph is highly connected. However, the improvement is limited and the computation time requirement is high.

Plesnik [45] partitions a graph into a spanning graph F and a number of induced subgraphs $(V_1, V_2, V_3, ..., V_k)$. By estimating the chromatic number of the spanning graph F through the same partitioning technique, the chromatic number of graph G is computed as the sum of the chromatic numbers of the

first f subgraphs (i.e, $g_1 + g_2 + g_3 + ... + g_f$), where g_i is the chromatic number for subgraph V_i and f is the chromatic number for spanning graph F. The partitioning technique is not clearly stated in the paper and the performance of the algorithm is not available.

Other researchers have proposed factoring techniques for coloring [18,45]. In these techniques, a graph is decomposed into factors $F_1, F_2, F_3, ..., F_j$ (F's are subgraphs). The chromatic number is estimated as the sum of the chromatic numbers of the subgraphs. However, good factoring techniques usually require large amount of computing time.

Several other algorithms are available and can be found in [5,31,36,40].

All the graph coloring algorithms mentioned above share a common feature: the algorithms are rather complex. Although Welsh and Powell's method is superior to others in terms of simplicity and computation time requirement [5], it predicts the chromatic number poorly when the graph is highly connected.

Graph coloring algorithms have been used in timetable design of examinations [61]. The events scheduled are examinations, each of which requires a period of time. The problem can be transformed into a graph coloring problem by representing each examination with a vertex of a graph and drawing an edge between the vertices if the corresponding examinations can not take place concurrently. The objective is to find the minimal number of periods required to accommodate all the examinations. Newman-Wolfe [43] uses edge coloring to schedule communication requests statically in multistage interconnection networks, where I/O ports and connection lines are represented by vertices and edges of a graph, respectively. Edges assigned the same color are switched on to allow communications to go through simultaneously.

Many people recognize the usefulness of the graph coloring algorithms, but because of the intractability of the graph coloring problem, few realistic applications have been proposed so far. Most approaches are static in nature, i.e., the participating events and constraints remain unchanged during the entire period of processing. The designation of vertices as processors in multiprocessor scheduling and the representation of switch boxes by vertices in multistage interconnection networks are examples of a static scheduling scheme which does not fully reflect the dynamic nature of request contention in a computer system and is therefore unable to optimize resource utilization. The approach proposed in this dissertation is to schedule events dynamically through graph traversal algorithms. The detailed description will be given in Chapter 4. This dissertation presents three graph traversal algorithms which give good estimations of the chromatic number of a graph in O(E) time where E is the number of edges of the graph. They can be applied to speed up the process of distinguishing conflicting and non-conflicting requests. However, due to the non-uniform ending of the requests, the subnetworks assigned to the processes that finish early will be idle. The problem can be solved by fragmenting the messages of the communication requests and by using a fast coin-changing algorithm to schedule them for execution.

2.4 Coin-changing Algorithms

In distributed processing, each individual processor needs to perform tasks independently and cooperate with other processors if necessary. Many of the computing tasks require cooperation among processors, and thus result in heavy loads on networks. Performance of the system degrades because of network contentions. The approach of using a graph coloring algorithm solves the contention problem and allows for parallel communication. What remains is the bus idling problem, which can

be solved by a coin-changing algorithm. The bus idling problem is due to the fact that the communication processes that are allowed to proceed in the partitioned subnetworks take different time durations, and the subnetworks that are released by the processes of shorter durations can not be reused until all the processes are completed.

The coin-changing problem can be defined as follows. For a given set of coin types $W = (W_1, W_2, ..., W_n)$, the goal is to meet a given total T value with a minimum number of coins. It is assumed that the supply for each type of coin is unlimited. Mathematically, the problem can be formulated as follows:

Minimize C = $\Sigma X_i \quad i = 1, 2, ..., n$

subject to $\Sigma W_i X_i = T$ where X_i is a non-negative integer.

The coin-changing problem is a simplified version of the integer programming problem. The existing work on this topic can be categorized as follows:

1. Exact algorithms: Researchers look for algorithms that can find optimal solutions for any arbitrary set of coin types. Chang and Gill [8] provides a recursive algorithmic solution which requires a considerable amount of computation time. Wright [62] proposes a solution using dynamic programming techniques to simplify the computation task. Both algorithms find optimal solutions. However, the computation time required makes the algorithms of little practical value.

2. The Greedy algorithm [8]: The Greedy algorithm is simple and fast. It takes as many of the largest coins as possible and then as many of the second largest coins as possible, etc. The procedure below implements the Greedy algorithm:

Procedure Coin – Changing (W, X, T);

(* W = (W_1, W_2, W_3, ... , W_n) *is the set of coin types*;

X = (X_1, X_2, X_3, ..., X_n) *is the representation*

of T *with respect to* W *)

begin

For i := n *to* 1 *do*

begin

X_i := *MOD* (T, W_i)

$T := T - X_i * W_i$

end;

end;

However, the algorithm does not always find optimal solutions. In some cases, the algorithm does not find a solution even though one exists. For instance, when the set of coin types is W = (5, 11, 15, 20) and the total is T = 30, the Greedy algorithm would return the representation X = (2, 0, 0, 1) instead of the optimal representation Y = (0, 0, 2, 0). If the total is T = 26, the Greedy algorithm would not return any representation even though an optimal representation, Y = (0, 1, 1, 0), exists. In this category, researchers are trying to determine if a set of coin types $W = (W_1, W_2, W_3, ..., W_n)$ will yield optimal solutions when the Greedy algorithm is applied as discussed in [7,8,58]. For instance, in Chang and Gill's paper, a range of values of T (the sum of the values of coin types) $(W_3, W_n(W_nW_{n-1} + W_n - 3W_{n-1})/(W_n - W_{n-1}))$ has to be checked for optimality with a given set of coin types W. Magazine et. al [34] provide methods that would check fewer values of T.

2.5 Summary

In order to meet the demand of a high performance local area network, both the graph coloring algorithms and coin-changing algorithms are applied in a dynamically partitionable bus network to improve the network performance. In this section, we have presented the survey of related work including partitionable bus networks, graph coloring algorithms, and coin-changing algorithms. The application, analysis, and performance evaluation of the graph coloring algorithms and the coin-changing algorithms will be presented in the next three chapters.

CHAPTER 3
GRAPH COLORING ALGORITHMS

The graph coloring algorithms are the foundation of this research. Three fast graph traversal algorithms which give good estimate of the chromatic number of a graph are presented. This chapter is organized as follows: Section 3.1 delineates the algorithms including the static, dynamic, and heuristic algorithms. Section 3.2 evaluates the performance of the dynamic graph traversal algorithm. Section 3.3 shows the results of the performance evaluation.

3.1 The Graph Traversal Algorithms

The algorithms to be presented below assign colors to the vertices of a graph through graph traversals. Each traversal results in some colored vertices and "thrown-away" vertices. Through repeated traversals of the "thrown-away" vertices, all vertices are assigned with colors.

3.1.1 Static Algorithm

Given a simple graph, the following steps are taken to color the vertices as illustrated in Figure 3.1.

1. Arbitrarily pick one vertex in the graph and label it as 1.

2. Assign a count to its adjacent vertices with a value that is one greater than the count of the vertex being traversed and mark the corresponding edges. Repeat this step on the new adjacent vertices until all edges are marked. If an adjacent vertex already has a count, a new count is assigned to it. Thus, vertices may have multiple counts.

3. Examine vertices with multiple counts, if the sum of any two counts associated with a vertex is odd, discard the vertex. For the remaining vertices, assign color 1 to those vertices whose sum of the counts is even. Assign color 2 to the rest of the vertices. This step colors the graph with two colors (color 1 and color 2) and discards some vertices.

4. The thrown-away vertices form a new graph which is then traversed starting from Step 1. The algorithm terminates if no thrown-away vertex remains.

Figure 3.1 shows an example of the graph traversal algorithm. In Figure 3.1a, vertex A is arbitrarily picked and is labeled as 1. The adjacent vertices of vertex A (i.e., vertex B and vertex C) are labeled as 2 as shown in Figure 3.1b. Repeat the same procedure: vertices D, E, and F are labeled as 3 in Figure 3.1c. Finally, in Figure 3.1d, vertex E is labeled as two 4's, due to its adjacency to vertex D and vertex F. So far, all the edges of the graph have been marked and vertex E has multiple counts. According to step 3 of the static graph traversal algorithm, vertices B and C are assigned with color 1, vertices A, D, and F are assigned with color 2, and vertex E is discarded.

Obviously, vertex E is assigned with color 3 in the next round of traversal. The approximate chromatic number of this graph produced by this algorithm is 3 which happens to be the actual chromatic number.

3.1.2 Dynamic Algorithm

The dynamic graph traversal algorithm is intended to improve the result of static algorithm in the following ways:

1. Reduce the number of thrown-away vertices.

2. Simplify the labeling technique.

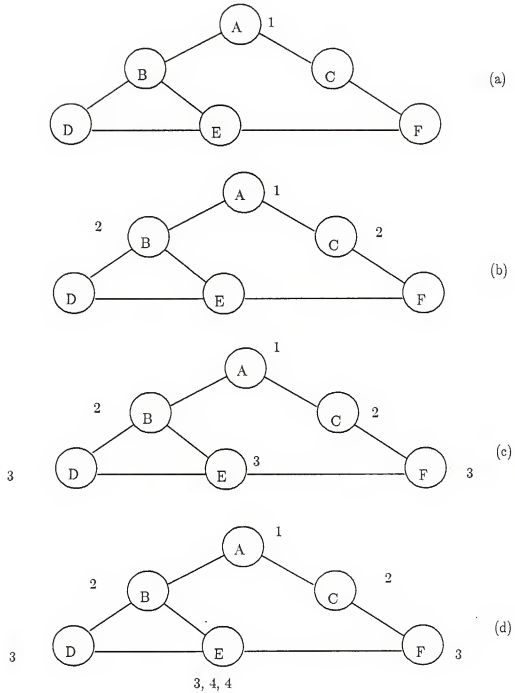


Figure 3.1. An Example of Graph Traversal Using the Static Graph Traversal Algorithm

3. Color the vertices of the graph "on the fly".

The dynamic graph traversal algorithm labels, colors, and throws away vertices as each vertex is traversed. A vertex whose sum of the counts is odd is discarded immediately to avoid further consideration.

The following steps form the algorithm:

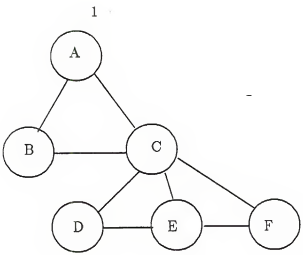
1. Arbitrarily pick one vertex and label it as 1. The vertex chosen is the front vertex.

2. Label the adjacent vertices of the front vertex in random order with the 1's complement value of the count of the front vertex. For each labeling of a new vertex, if the sum of any two counts is 1, discard the vertex right away. Repeat this procedure for each vertex adjacent to the front vertices being traversed.

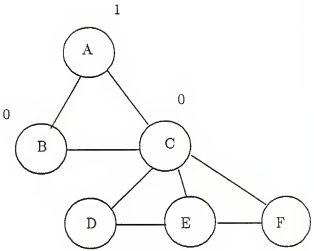
3. Designate the newly labeled vertices as the front vertices. Go to step 2 if there are still unlabeled vertices.

4. Go to step 1, if there are vertices that have been thrown away in step 2.

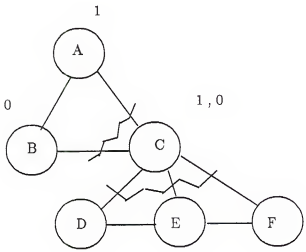
Figure 3.2 demonstrates the steps of the dynamic graph traversal. In Figure 3.2a, vertex A labeled as 1 is the front vertex. Vertices B and C which are adjacent to the front vertex A are labeled as 0 in Figure 3.2b (complement value of the count of the front vertex A). Note that the counts of each vertex are checked immediately to see if the vertex needs to be thrown away. At this point, vertices B and C are the new front vertices. In Figure 3.2c, vertex C is labeled as 1 and is thrown away because it is adjacent to vertex B. As a result, since there is no more front vertex to be processed, the algorithm would randomly pick a new starting vertex from the unlabeled vertices (i.e., vertices D or E) and label it as 1, as shown in Figure 3.2d. According to step 2, vertex E is labeled as 0. So far, all the vertices have been labeled and colored. The



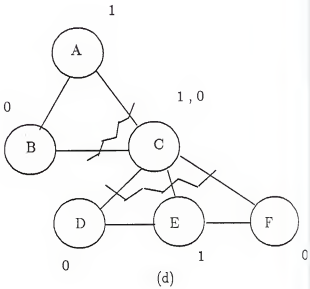
(a)



(b)



(c)



(d)

Figure 3.2. An Example of Graph Traversal Using the Dynamic Graph Traversal Algorithm

group of vertices with a count of 1 is colored with the first color, the group of vertices with a count of 0 is colored with the second color, and the vertex C is colored with the third color in the next round of traversal.

The advantages of this algorithm are obvious as compared to the static graph traversal algorithm. They are:

- Vertices are thrown away "on the fly" as they are traversed so that the edges associated with them do not have to be traversed as they do in the static algorithm.
- Fewer edges need to be traversed.
- Labeling vertices with 0's and 1's is simpler than counts.

In addition, the dynamic graph traversal algorithm has some mathematical properties which simplify the analysis of its performance. Details are given in Section 3.2.

3.1.3 Heuristic Algorithm

In some cases, both the static and dynamic algorithms will throw away a high percentage of vertices. Figure 3.3 gives an example. Suppose that vertex A is chosen as the front vertex using the dynamic algorithm and labeled as 1 and vertices B, C, D, and E are labeled as 0. If the labeling is in the direction of B–>C, B–>D, and B–>E in the next step, three vertices (C, D, and E) will be thrown away. On the other hand, if the labeling of vertices is in the opposite direction (C–>B, D–>B, and E–>B), only vertex B will be thrown away.

This motivates us to pursue some heuristics to further reduce the number of vertices that will be thrown away. Two heuristic rules are given below:

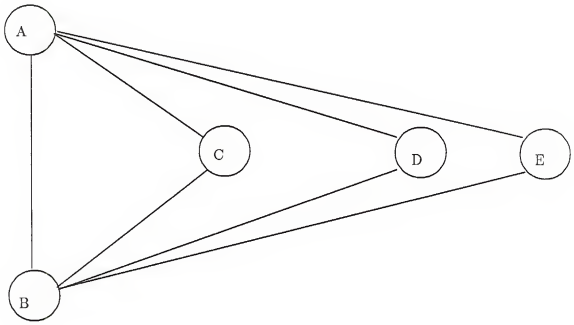


Figure 3.3. An Example of Graph Traversal Using the Heuristic Graph Traversal Algorithm

1. Pick the vertex with the largest degree (i.e., the largest number of edges) as the starting vertex. The idea is to expand the set of front vertices as soon as possible.

2. In exploring the adjacent vertices of a front vertex, start with the vertex of the smallest degree. The justification for this rule is to throw away the vertices that break up most odd cycles as soon as possible. The odd cycles are the ones with an odd number of vertices.

If we incorporate the heuristics proposed above to the dynamic graph traversal algorithm, the number of thrown-away vertices can be significantly reduced. The example in Figure 3.3 throws away only one vertex when the heuristic graph traversal algorithm is used.

3.2 Analysis of the Dynamic Graph Traversal Algorithm

The dynamic graph traversal algorithm has some mathematical properties which yield a simplier analysis of the bound of the approximate chromatic number. First, we define some theorems and related terms.

Definition 3.2.1 Bipartite graph: A graph G (V, E) is a bipartite graph if there exist subsets V1 and V2 of V where V1 $\uplus^1$ *V2 = V, such that each edge of the graph (i.e., e* $\in$ *E) has one endpoint in V1 and the other in V2. A graph is a bipartite graph if and only if it is two-colorable.*

Corollary 3.2.1 The dynamic graph traversal algorithm is an efficient way of determining if a graph is two-colorable (bipartite). Applying the dynamic traversal algorithm to a graph G, if a vertex is thrown away in the middle of traversal, the graph is not two-colorable.

[1]union disjoint set

Corollary 3.2.2 Each traversal of the dynamic graph traversal algorithm partitions the vertices of the graph into three groups of vertices: vertices with color 1, vertices with color 2, and the thrown-away vertices.

Definition 3.2.2 Independent set: A set of vertices is said to be independent if no two of its vertices are adjacent to each other.

Definition 3.2.3 Maximal independent set: An independent set I is maximal if no other independent sets contain I in graph G.

Theorem 3.2.1 For a connected graph, each traversal of the dynamic algorithm generates two maximal independent sets (i.e., the set of vertices with color 1 (set 1) and the set of vertices with color 2 (set 2)). It should be noted that a maximal independent set is not the same as a maximum independent set.

Proof of Theorem 3.2.1 The claim that the set of vertices with color 1 is a maximal independent set is based on the following argument. No vertices in set 2 can be added to set 1 without breaking its independence, since each vertex in set 2 is adjacent to at least one vertex in set 1. Furthermore, no vertex in the set of thrown-away vertices (set 3) can be added to set 1, since each vertex in set 3 is adjacent to at least one vertex in set 1 and one vertex in set 2. As a result, set 1 must be a maximal independent set. The same argument can be used to prove that set 2 is also a maximal independent set.

Theorem 3.2.2 The dynamic graph traversal algorithm produces the exact chromatic number 2 for a bipartite graph.

Proof of Theorem 3.2.2 Without loss of generality, it can be assumed that the graph G is connected. (If the graph is not connected, the algorithm simply treats each

component separately.) For a connected bipartite graph, the number of maximal independent sets is two which can be determined by the dynamic graph traversal algorithm in the first traversal of the graph (Theorem 3.2.1). No vertices are thrown away. As a result, the algorithm generates the exact chromatic number for a bipartite graph.

Definition 3.2.4 k-partite graph: A graph whose vertex set can be partitioned as V = V1 ⊎ V2 ⊎...⊎ Vk such that there exists no edge between a pair of vertices located in the same partitioned set.

Definition 3.2.5 Complete k-partite graph: A k-partite graph is complete if there exists an edge between each pair of vertices located in distinct partitioned sets.

Theorem 3.2.3 The dynamic graph traversal algorithm always produces the exact chromatic number k for a complete k-partite graph.

Proof of Theorem 3.2.3 Let G(V, E) be a complete k-partite graph with vertex partitions V1, V2,...,and Vk. Each edge of this graph has its endpoints in two distinct partitions (see Figure 3.4a for an example of k-partite graph). Without loss of generality, a vertex s1 in V1 is picked as the starting vertex and is assigned with a count of 1. In Figure 3.4b, the adjacent vertices of the starting vertices s1 (all the vertices not in V1) are assigned a count of 0 according to the dynamic graph traversal algorithm. Now, all the vertices not in V1 are the front vertices. In the next step, pick a vertex from the set of front vertices, say, s2 in V2 for expansion. The adjacent vertices of s2 are given a count of 1 as shown in Figure 3.4c. At this step, all the vertices in V3 through Vk are thrown away because the sum of two counts is odd and the vertices in V1 and V2 are assigned with colors 1 and 2, respectively. Thus, a new (k-2)-partite graph is formed with vertex partition V3, V4,...,and Vk. By repeating the same steps, the dynamic graph traversal algorithm uses k colors for the k-partite graph.

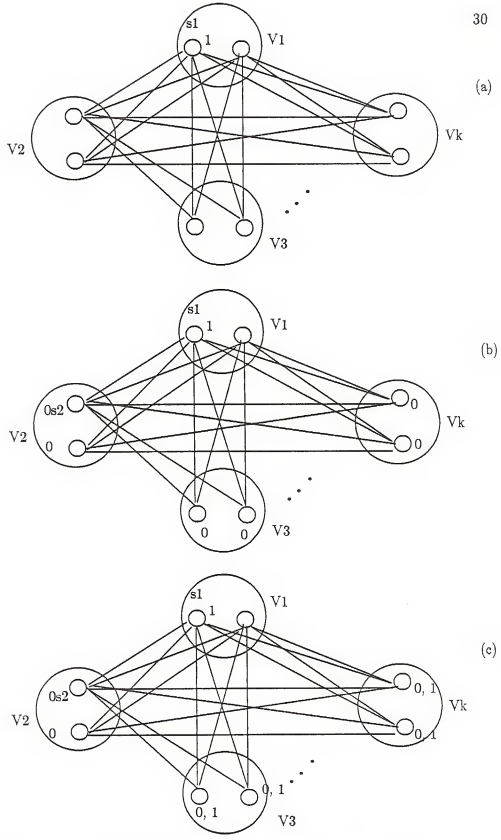


Figure 3.4. An Example of Graph Traversal on a K-partite Graph

Definition 3.2.6 Random graph model: A random graph model M(| V |, q) is defined in terms of the number of vertices of a graph (| V |) and the probability (q) that an edge exists between any distinct pair of vertices, where $0 \leq q \leq 1$. This model consists of a set of graphs, each of which has | V | vertices and its edges are determined by a random number generator. For a given q value, if a random number is less than q, then the edge between two vertices exists. A set of graphs is generated for the model in this manner.

The random graph model is used in the performance evaluation of the dynamic graph traversal algorithm, since it generates all possible graphs with equal probability.

Definition 3.2.7 Clique: A clique here means a maximal complete subgraph of a graph.

Theorem 3.2.4 The expected number of maximal independent sets of a random graph containing a fixed vertex has been derived in [35,59] as follows:

$$E_1 = \sum_{d=1}^{|V|} \binom{|V-1|}{d-1} (1-q)^{d(d-1)/2} (1-(1-q)^d)^{|V|-d}$$

where

E_1: the expected number of maximal independent sets of a random graph model containing a fixed vertex

| V | : number of vertices in graph G

q: the probability that an edge exists between any pair of vertices in a random graph

The calculation of the number of maximal independent sets for a random graph model M(| V |, q) is done by counting the number of cliques and replacing q with (1-q), since the complement of a maximal independent set is a clique [15]. The counting of cliques can be explained as follows:

1. The size of cliques ranges from 1 to | V |.
2. For each size of cliques (d), there will be $\binom{d}{2}$ edges and the probability is q raised to the $\binom{d}{2}$th power.
3. If a vertex can be added to a clique, it has to be fully connected to the vertices in the clique. The probability that this situation happens is q raised to the dth power. The probability that a vertex can not be added to a clique is therefore (1 - q^d). Since there are (| V | - d) vertices not in the clique, (1 - q^d) has to be raised to the (| V | - d)th power.
4. The product of the probabilities derived in items 2 and 3 is the probability that a clique with d vertices exists.
5. Since a fixed vertex must be included in the clique, the number of combinations for a clique of size d is
$$\binom{|V|-1}{d-1}$$
6. The product of the terms derived in items 4 and 5 is the expected number of cliques of d vertices in a random graph model with | V | vertices under the condition that a fixed vertex must be included in the clique.
7. $\overline{E_1}$ (the expected number of cliques) is calculated by taking the summation of the expected numbers of cliques for different sizes of cliques.

$$\overline{E_1} = \sum_{d=1}^{|V|} \binom{|V-1|}{d-1} q^{d(d-1)/2}(1-q^d)^{|V|-d}$$

8. The expected number of maximal independent sets is obtained by replacing q with (1-q) in $\overline{E_1}$, since cliques and maximal independent sets complement each other.

$$E_1 = \sum_{d=1}^{|V|} \binom{|V-1|}{d-1} (1-q)^{d(d-1)/2}(1-(1-q)^d)^{|V|-d}$$

Theorem 3.2.5 *The expected number of maximal independent sets of a random graph model is*

$$E_2 = \sum_{d=1}^{|V|} \binom{|V|}{d} (1-q)^{d(d-1)/2}(1-(1-q)^d)^{|V|-d}$$

where

E_2: the expected number of maximal independent sets of a random graph model

The calculation is similar to Theorem 3.2.4.

Definition 3.2.8 *Color class: In a colored graph, the set of vertices with the same color forms a color class.*

Definition 3.2.9 *Expected chromatic number (ECN): The expected value of the chromatic number for a random graph model is calculated by taking the summation of the product of each possible chromatic number and its probability.*

It should be noted that the closed form for the expected chromatic number is not known at this moment. In this section, the approximate value of the expected chromatic number which can be obtained by applying the dynamic graph traversal algorithm to a random graph is computed.

Methods for coloring random graphs have been proposed by several researchers [4,14,24,32]. Various constraints on the random graphs, such as small graphs, dense graphs, etc., were introduced in their algorithms. This work considers general random graphs without constraints. In the rest of this section, the theoretical basis for evaluating the performance of the dynamic graph traversal algorithm on general random graphs is provided.

Given a random graph, the assignment of colors to its vertices is done by applying the dynamic graph traversal algorithm repeatedly to the random graph until each vertex has been assigned a color. Since each traversal produces two maximal independent sets, and each maximal independent set forms a color class, the number of maximal independent sets produced by the algorithm is equal to the approximate value of the expected chromatic number of the random graph model defined by $\mid V \mid$ and q.

The approximate value of the expected chromatic number can be calculated as follows:

Step 1. Calculate the expected size of the maximal independent sets (ESMIS).

The expected size of maximal independent sets (ESMIS) is equal to the sum of the number of maximal independent sets with d vertices times d vertices (where d may go from 1 to $\mid V \mid$) and divided by the total number of maximal independent sets, i.e.,

$$ESMIS = \frac{\sum_{d=1}^{|V|} d \begin{pmatrix} |V| \\ d \end{pmatrix} (1-q)^{d(d-1)/2}(1-(1-q)^d)^{|V|-d}}{\sum_{d=1}^{|V|} \begin{pmatrix} |V| \\ d \end{pmatrix} (1-q)^{d(d-1)/2}(1-(1-q)^d)^{|V|-d}}$$

The numerator can be simplified.

$$\sum_{d=1}^{|V|} d \begin{pmatrix} |V| \\ d \end{pmatrix} (1-q)^{d(d-1)/2}(1-(1-q)^d)^{|V|-d}$$

$$= \sum_{d=1}^{|V|} \frac{|V|!}{(d-1)!(|V|-d)!}(1-q)^{d(d-1)/2}(1-(1-q)^d)^{|V|-d}$$

$$= |V| \sum_{d=1}^{|V|} \frac{(|V|-1)!}{(d-1)!(|V|-1-(d-1))!}(1-q)^{d(d-1)/2}(1-(1-q)^d)^{|V|-d}$$

$$= E_1 |V|$$

Thus,

$$ESMIS = \frac{E_1}{E_2} |V|$$

where E_1 and E_2 are the expected number of maximal independent sets containing a fixed vertex and the expected number of maximal independent sets, respectively.

Step 2. The approximate value of the expected number of maximal independent sets is calculated by dividing the number of vertices of the graph by the expected size of maximal independent sets, i.e.,

$$AECN = \frac{|\ V\ |}{ESMIS} = \frac{E_2}{E_1}$$

where

AECN = approximate value of the expected chromatic number

$|\ V\ |$ = number of vertices of the random graph

ESMIS = expected size of maximal independent sets

The dynamic graph traversal algorithm partitions a random graph into a number of maximal independent sets which are picked from a pool of all possible maximal independent sets for the random graph. Depending on the patterns of random graphs, different sets of maximal independent sets will be chosen. Even for random graphs with the same pattern, the chosen maximal independent sets may vary, since the algorithm picks the starting vertex arbitrarily when traversing the graph and the order of exploring the adjacent vertices of a front vertex is random. Since each maximal independent set of the pool of maximal independent sets has the same probability to be chosen, and the expected size of the maximal independent sets in the pool can be computed as a function of ($|\ V\ |$) and (q), ($|\ V\ |$ / ESMIS) is the expected number of maximal independent sets that will be produced by the dynamic graph traversal algorithm. If each maximal independent set is a color class, the expected number of maximal independent sets is the approximate value for the expected chromatic number.

As an example, for a random graph with 50 vertices and the probability that an edge exists between any pair of vertices is 0.5, the following values can be derived using the above equations.

$E_1 = 98.78$

$E_2 = 963.50$

ESMIS = 5.14

AECN = 9.7

3.3 Comparison of Results

In this section, both the simulation results obtained by the method that orders vertices and uses a similarity matrix [61] and the analytical results of the dynamic graph traversal algorithm are provided. The results are shown in Table 3.1.

Table 3.1. The Approximate Values of the Expected Chromatic Number by Wood's Method and the Dynamic Graph Traversal Algorithm

$\mid V \mid$	q	α	β
20	0.25	3 - 5	3.11
20	0.50	5 - 8	5.11
20	0.75	8 - 10	7.89
50	0.25	6 - 8	5.43
50	0.50	10 - 13	9.75
50	0.75	16 - 20	15.73
100	0.25	10 - 13	8.78
100	0.50	18 - 22	16.52
100	0.75	28 - 33	27.63

In the table, $\mid V \mid$ and q are the number of vertices and the probability that an edge exists between a pair of vertices, respectively. α is the range in which most approximate chromatic numbers produced by Wood's algorithm fall, and β is the approximate value of the expected chromatic number computed using our formula. As shown in the table, β stays in the low end of α for various $\mid V \mid$'s and q's. As can be seen, the dynamic graph traversal algorithm produces better results. It should be noted that the comparison is based on the analytical results of the dynamic algorithm

and the simulation results of Wood's method. Since the domains of samples are not exactly the same, the difference between α and β may not be as large as it is shown in Table 3.1. The comparison based on simulation results will be included in the future work.

CHAPTER 4
DPBN AND GRAPH COLORING ALGORITHM

One way to increase bus capacity is to physically partition the bus, allowing parallel communications in the fragmented segments. Figure 4.1 shows the architecture of a partitionable bus [3,54] which consists of a number of stations connected to a shared communication network in a multipoint configuration. A number of switches are used to physically partition the bus into clusters, each of which contains a number of adjacent stations. For example, by closing all switches except the kth switch, two clusters are formed; one contains stations 1 to k and the other contains k+1 to n stations where n is the total number of stations in the network. Communications among stations in one cluster can be carried out in parallel with the communications in another cluster. The control of the switches can be either centralized or distributed. In a centralized control, the control computer sets the switches for partitioning or connecting the bus. In a decentralized control, an individual station turns the switch on and off when receiving a command from the control computer.

The above idea of using a dynamically partitionable bus will be useful if there is an efficient algorithm to distinguish conflicting and non-conflicting communication requests so that the network can be properly partitioned to allow a maximum of non-conflicting requests to be processed in parallel.

In the context of a partitionable bus network, two communication requests are said to be in conflict if the range of adjacent stations defined by the sender and the receiver of one request overlaps with the range defined by the sender and the receiver of the other request. For example, if station 1 wants to send a message to station

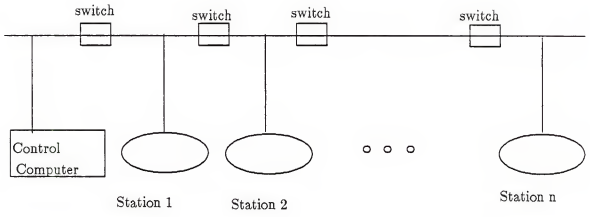


Figure 4.1. The Architecture of a Partitionable Bus Network

4 and station 3 wants to send a message to station 5 at the same time, these two requests would be in conflict since the range 1 to 4 overlaps with the range 3 to 5 and the bus can not be partitioned to allow simultaneous transmissions of these requests. Graph traversal algorithms can be applied to efficiently distinguish conflicting and non-conflicting requests.

This chapter is organized as follows: Section 4.1 describes the application of the dynamic graph traversal algorithm. Section 4.2 delineates the bus partitioning technique. Section 4.3 presents the hardware design of the dynamic graph traversal algorithm. Section 4.4 gives the results of a performance evaluation.

4.1 A Graph Traversal Algorithm and Its Application

In Section 3.1, three graph traversal algorithms: (static, dynamic, and heuristic graph traversal algorithms) were presented. Each traversal of the algorithm results in some colored vertices and "thrown-away" vertices. Through repeated traversals of the "thrown-away" vertices, all vertices are assigned with colors. The dynamic graph traversal algorithm has several advantages over the static traversal algorithms and is suitable for real-time applications.

The application of the dynamic algorithm in a computer system is as follows. If each vertex of the graph (Figure 3.2) represents an event of a computer system, the events corresponding to vertices B, D, and F can be processed in parallel, as can the events corresponding to vertices A and E. The corresponding event of vertex C, which is thrown away in the graph traversal, will join the next group of processes to compete for a shared system resource.

As a result of separating conflicting and non-conflicting events, the system performance can be enhanced. If the vertices in Figure 3.2 represent communication requests of a single shared bus, the system throughput can almost be doubled. In the

case of a very busy network, the increase in system throughput will be even higher. The traversal time of this algorithm is negligible since it is a very simple algorithm and the time can most probably be overlapped with the communication processes.

4.2 Dynamic Bus Partitioning Technique

This section shows how the graph coloring algorithm can be applied in a dynamically partitionable bus network. The following steps are required to construct a graph for a partitionable bus network.

1. Represent each communication request from one station to another by a vertex in the graph.

2. Determine the conflicts among the communication requests. If there is a conflict between two communication requests as defined in the beginning of this chapter, draw an edge between the corresponding vertices.

3. Apply the graph traversal algorithm to the constructed graph. (Note: Some of the vertices may be thrown away as described in the graph traversal algorithm).

4. At time t1, all the vertices (representing communication requests) with color 1 are allowed to perform communication in parallel in different clusters of stations which are formed by physically partitioning the bus. Then, at time t2, all the vertices with color 2 are allowed to perform communication in parallel by repartitioning the bus. The vertices which have been thrown away at this traversal will join the next group of communication requests.

Tables 4.1 and 4.2 and Figures 4.2 and 4.3 illustrate the steps of the bus partitioning technique. Five communication requests are collected in a communication request table (Table 4.1). Each request is represented by a vertex in the graph; for

example, the communication request from station 1 to station 2 is denoted by vertex A. Table 4.2 shows the conflicts among the communication requests. A "C" in a table entry indicates that there is a conflict between the request on the row and the request on the column. The graph representation of the communication requests shown in Figure 4.2 is constructed according to the conflicting events shown in Table 4.2. In Figure 4.2, communication requests are represented by vertices (A, B, C, D, and E) and edges are drawn to indicate their conflicts. Figure 4.3 shows the result of the graph traversal.

Table 4.1. A Graph Representation of a Group of Communication Requests

From station	To station	Graph representation (vertex)
1	2	A
2	4	B
3	4	C
4	5	D
5	6	E

Table 4.2. The Table of Conflicts among Communication Requests

	1 - 2	2 - 4	3 - 4	4 - 6	5 - 6
1 - 2		C			
2 - 4	C		C	C	
3 - 4		C		C	
4 - 5		C	C		C
5 - 6				C	

In this example, vertices A and C are assigned color 1, vertices B and E are assigned color 2, and vertex D is thrown away. At time t1, the communication request from station 1 to station 2 and the communication request from station 3 to station 4 can be done in parallel by opening the switch between station 2 and

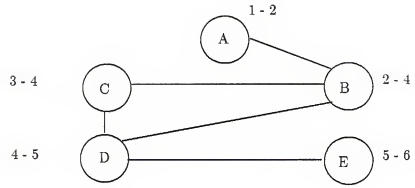


Figure 4.2. A Graph Representation of Conflicts and Requests

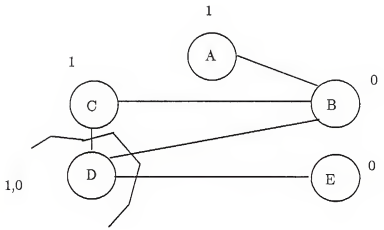


Figure 4.3. The Graph Traversal of the Constructed Graph

station 3 and closing all other switches. At time t2, the communication request from station 2 to station 4 and the communication request from station 5 to station 6 can be done in parallel by resetting the switches properly. The communication request from station 4 to station 5, which is represented by vertex D, will join the next group of communication requests. In a system that uses a dynamically partitionable bus, the bus can be properly partitioned to allow parallel communication to take place at different times. Thus, system throughput can be dramatically increased by using the graph traversal algorithm in conjunction with the partitionable network. The example given above demonstrates our approach and algorithm. The results of a simulation study of DPBN will be given in Section 4.4.

There are several notable features in our approach to solving contention problems in a computer system.

1. Flexible Representation: Vertices can be used to represent any type of requests for any resource in a computer system. Any resource contention problem can be transformed into a graph coloring problem. For instance, in database management, vertices can denote database operations (JOIN, SELECT, AND PROJECT), and edges can represent memory access conflicts among database operations. In multiple bus networks, partitionable networks, and multistage interconnection networks, vertices and edges can represent communication requests and conflicts in using communication network(s), respectively. This flexibility in problem representation allows for a broad range of applications of the presented algorithm. As long as a system has multiple processes competing for limited resources, the proposed approach can be applied.

2. Fast Running Time: It is obvious that the complexity of the graph traversal algorithm is O(E), where E is the number of edges of the graph, since each

edge is processed once only. The required memory space for traversing the graph is small, and the traversal can overlap with packet communication and data processing to reduce the overhead involved (i.e., while processes are being processed in one time frame, the algorithm is used to traverse the graph for the next group of requests).

3. Multiple Ways of Interpreting the Colors Assigned to the Vertices. Depending on the applications, the colors assigned to the vertices of the graph can be interpreted in a variety of ways. For instance, the colors assigned to the vertices of the graph can be interpreted as time frames in partitionable bus networks and database query processing. The corresponding requests of the vertices assigned with different colors must be performed at different time frames. However, in multiple bus networks, the colors can be interpreted as buses. The corresponding requests of the vertices assigned with different colors can be carried out in different buses at the same time frame.

4.3 Hardware Design of A Graph Traversal Unit

In a real-time environment, many requests for a resource arrive at the same time and a very fast scheduling decision needs to be made. The proposed algorithm may not meet the time requirement if it is implemented in software. In that case, a hardware implementation of the algorithm would be needed. In this section, a design of the graph traversal unit is presented.

4.3.1 Architecture of the Graph Traversal Unit

The graph traversal unit is a specialized hardware with the capability of distinguishing conflicting and non-conflicting communication requests by assigning either a "0" or a "1" to each communication request. It is connected to the control unit

of the control computer and can operate in parallel with the normal operation of the control computer, which off-loads communication requests to the graph traversal unit. As shown in Figure 4.1, the control computer and stations are suspended off the partitionable bus. Figure 4.4 shows the detailed organization of the control computer, which consists of the usual components of a general purpose computer including disks, memory, DMA, control unit, and additionally a graph traversal unit. In general, the system works as follows:

1. The control computer collects communication requests from the stations and produces an adjacency matrix such as the one illustrated in Section 4.2. A communication request is composed of two numbers, the tag of the sender and the tag of the receiver. For each pair of communication requests, the corresponding tags are used to determine if there is a conflict. A conflict occurs if the range of consecutive stations represented by a pair of numbers overlaps with the range represented by another pair as shown in the following procedure.

Procedure For Determining Conflict;
(**Let the Tags for a pair of communication requests be*
(T_{1a}, T_{1b}) *and* (T_{2a}, T_{2b}), *respectively.*
*Each pair of numbers represents a range of consecutive stations.**)
begin
If$(T_{1a} < T_{2a}$ *and* $T_{1a} < T_{2b}$ *and* $T_{1b} < T_{2a}$ *and* $T_{1b} < T_{2b})$
then return no conflict
elseif$(T_{1a} > T_{2a}$ *and* $T_{1a} > T_{2b}$ *and*$T_{1b} > T_{2a}$ *and* $T_{1b} > T_{2b})$
then return no conflict
else

return conflict

end;

2. The control computer maps the adjacency matrix to the RAM of the graph traversal unit and activates the unit.

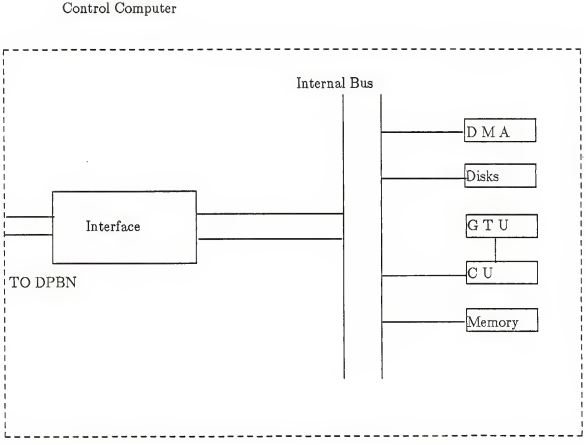
3. The graph traversal unit driven by an internal controller returns the results of graph traversal to the control computer, which sets switches on/off to partition the bus accordingly to allow messages to go through in parallel.

It should be noted that the determination of conflicts among communication requests can be done by either the CPU of the control computer or the graph traversal unit. If the graph traversal unit determines the conflicts, an additional logic unit would be required.

4.3.2 Input and Output of the Graph Traversal Unit

The input to the graph traversal unit is nothing but the adjacency matrix. The adjacency matrix is a RAM of size 128 x 128, which is essentially a two dimensional array with each element indicating the adjacency of the vertex on the row and the vertex on the column. A conflict between any two communication requests is indicated by placing a 1 in the corresponding position in the adjacency matrix. Depending on the number of communication requests being processed by the graph traversal unit, the size of RAM can be changed to suit system's need. The number 128 was chosen arbitrarily for illustration purposes. It is a system design parameter.

The results of a graph traversal are stored in two 128 x 1 registers (status matrix) which use two bits (one bit from each register) to show the status of each vertex: colored with the first color, colored with the second color, and thrown away for the next traversal. In addition to the status matrix, a status code is used to indicate the



Note: G T U stands for graph traversal unit

Figure 4.4. The Detailed Organization of the Control Computer of the Partitionable Bus Network (DPBN)

status of the graph traversal unit. When the results of traversal are ready, a signal is sent to the control computer. On the other hand, the control computer may issue one of the three signals to specify the external functions to be executed by the graph traversal unit. The external functions are as follows:

1. Clear: Before loading the adjacency matrix, the RAM is cleared.

2. Load: The control computer off-loads the graph traversal unit with the adjacency matrix.

3. Go: The graph traversal unit is activated. The controller of the graph traversal unit starts executing micro-instructions and produces results. The external function Go is composed of a sequence of logical operations.

The following example illustrates the traversal of the graph in Figure 4.2 by the GTU. To simplify the illustration, the size of the RAM is reduced to 8 x 8 and the size of the registers is reduced to 8 bits.

1. Graph traversal unit is enabled.

2. The RAM is loaded with the adjacency matrix. As shown in Table 4.3, each address of the RAM (addresses 000 through 111) represents a communication request. For instance, the address 000 represents a communication request from station 1 to station 2 (vertex A), and the adjacent vertices of vertex A are the corresponding vertices of the bits with 1 in the row of address 000. It is obvious that vertex B is the only adjacent vertex of vertex A (address 000) since the second bit of the row 000 is 1 and all other bits are 0's. It should be noted that only the upper triangle of the RAM is filled with 1's if the corresponding communication requests are in conflict. For example, the

conflict between vertex A (address 000) and vertex B (address 001) is indicated by putting a 1 in the entry (1, 2) instead of in the entry (2, 1).

Table 4.3. The Adjacency Matrix Stored in the RAM

000 (vertex A)	0	1	0	0	0	0	0	0
001 (vertex B)	0	0	1	1	0	0	0	0
010 (vertex C)	0	0	0	1	0	0	0	0
011 (vertex D)	0	0	0	0	1	0	0	0
100 (vertex E)	0	0	0	0	0	0	0	0
101 (unused)	0	0	0	0	0	0	0	0
110 (unused)	0	0	0	0	0	0	0	0
111 (unused)	0	0	0	0	0	0	0	0

3. There are eight working registers ($W_0, W_1, ..., W_7$) used to support the hardware graph traversal algorithm. They are initialized with values as shown in Table 4.4. Each bit of a working register represents the status of the corresponding

Table 4.4. The Initial Values of the Registers ($W_0, W_1, W_2, ..., W_7$)

W_0	0	0	0	0	0	0	0	0
W_1	0	0	0	0	0	0	0	0
W_2	0	0	0	0	0	0	0	0
W_3	1	0	0	0	0	0	0	0
W_4	0	0	0	0	0	0	0	0
W_5	1	1	1	1	1	1	1	1
W_6	1	0	0	0	0	0	0	0
W_7	0	0	0	0	0	0	0	0

vertex. For instance, the first bit of a working register represents the status of vertex A, the second bit represents the status of vertex B, etc. The purposes of these working registers and their relationship to the dynamic graph traversal algorithm are explained below:

In the dynamic algorithm, the starting vertex is randomly chosen. However, the starting vertex must not be an isolated vertex. W_5 is used for searching for a starting vertex. This is done by searching the RAM row by row. The corresponding vertex of a row that is not null can be designated as the starting vertex. In this example, vertex A (address 000) is chosen as the starting vertex.

Once the starting vertex (front vertex) is chosen, the adjacent vertices of the front vertices need to be explored. W_0 stores the adjacent vertices of the front vertex. The register W_0 is an 8-bit register. Each bit indicates the status (adjacent or non-adjacent) of the corresponding vertex to the front vertex. For instance, the row of address 001 of the RAM is loaded into W_0 and the third and fourth bits of W_0 are 1's. It means that vertex C (represented by the third bit) and vertex D (represented by the fourth bit) are adjacent to vertex B (RAM address 001).

In the process of traversal, the adjacent vertices need to be assigned the current count. W_1 stores the current count that is being assigned to the vertices. The bits of W_1 are either all 1's or all 0's. After the expansion of each of the front vertices, the current count is replaced with a 2's complement value of the current count. In other words, the count is alternating between 1 and 0.

To record the adjacent vertices assigned with current count (either 0 or 1), W_2 and W_3 are used. W_2 stores the current count of the adjacent vertices. In each expansion of a front vertex, the adjacent vertices of the front vertex are assigned a current count stored in W_1. If the current count stored in W_1 is 1, the contents of W_0 are copied to W_2. This operation will assign a count of 1 to the bits of W_2 where the corresponding bits in W_0 are 1. If the current count stored in W_1 is 0, no count needs to be assigned, since W_2 is initialized to zero.

W_3 stores the history of counts (accumulative counts) assigned to each vertex. It should be noted that only one bit in W_3 is enough for recording the history of the counts of a vertex, since multiple 1's are the same as a single 1 and multiple 0's are the same as a single 0's in evaluating whether a vertex should be thrown away or what color should be assigned to the corresponding vertex. The current count assigned to W_2 is accumulated in W_3 after each expansion of a front vertex. In other words, W_3 contains all the counts which have been assigned to the vertices. This can be done by taking a logical "OR" operation between W_2 and W_3. The first bit of W_3 in this example is initialized to 1, since vertex A is designated as the starting vertex and has been assigned a count of 1.

While assigning the current count to each adjacent vertex, each vertex is checked to see if the condition of "thrown-away" (i.e., a vertex that has at least one count of 1 and at least one count of 0) is met. The results of checking of "thrown-away" are stored in W_4. The corresponding vertices of the bits with the value 1 in W_4 are "thrown-away" vertices. The "thrown-away" vertices can be determined by taking a logical "AND" operation between W_3 and W_7 (to be given below).

After each of the set of front vertices has been expanded, the adjacent vertices of the current front vertices become a new set of front vertices. W_6 stores the set of front vertices. The corresponding vertices of the bits with a value of 1 are the front vertices. In our example, the first bit of W_6 is initialized to 1, since vertex A is the starting vertex (front vertex).

In order to distinguish between a vertex assigned with a count of 0 and a new vertex (a vertex that has no count), W_7 is used to keep track of the vertices that

have been assigned a count of 0. A vertex that has been assigned a count of 0 is identified by putting a 1 in the corresponding bit of W_7. This can be achieved by taking a logical "OR"operation between W_0 and W_7. In this example, W_7 is null, since no vertices have been assigned a count of 0.

4. The adjacent vertices of the starting vertex (front vertex) are explored by loading the row of address 000 into W_0 ($W_0 \leftarrow$ RAM[MAR]). Here, MAR stands for the memory address register. The contents of W_0 are shown below.

$$W_0 \quad 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0$$

5. Keep track of the vertices that have been assigned a count of 0 ($W_7 \leftarrow W_0 \vee W_7$). When a front vertex is expanded, some vertices are newly traversed. The newly traversed vertices (indicated by the mark bits with 1's in W_0) are added to the set of vertices which have already been assigned a count of 0 (indicated by the mark bits with 1's in W_7) by taking a logical "OR" operation between W_0 and W_7. It should be noted that this step is necessary only when the current count is 0. The contents of W_7 are shown below.

$$W_7 \quad 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0$$

6. Establish a new set of front vertices and complement the count ($W_6 \leftarrow W_0$ and $W_1 \leftarrow\sim W_1$). W_1 contains the current count of the traversal. When a new set of front vertices is established, the count is replaced with the 2's complement of the original count as described in the algorithm. The contents of W_1 and W_6 are as follows:

$$W_1 \quad 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$$

$$W_6 \quad 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0$$

7. Encode each mark bit of W_6. The data stored in the encoded address of the RAM are read into W_0 ($W_0 \leftarrow$ RAM[MAR]). This explanation is very similar to that of step 1. Each of the front vertices is explored by loading the corresponding row of the RAM into W_0. The contents of W_0 are shown below.

$$W_0 \quad 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0$$

8. Assign the current count to each adjacent vertex ($W_2 \leftarrow W_0$). W_0 and W_1 contain the adjacent vertices of the front vertex and the current count, respectively. Since the current count is 1 this operation assigns the current count to the adjacent vertices of the front vertex. The contents of W_2 are now

$$W_2 \quad 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0$$

9. Accumulate the counts ($W_3 \leftarrow W_2 \vee W_3$). W_3 contains the accumulated counts and W_2 contains the count assigned to the adjacent vertices. The logical "OR" operation accumulates the counts. The contents of W_3 are now

$$W_3 \quad 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0$$

10. Check for "thrown-away" vertices ($W_4 \leftarrow W_3 \wedge W_7$). W_3 and W_7 keep track of the count of 1 and the count of 0 of the vertices, respectively. The logical "AND" operation determines the "thrown-away" vertices. The corresponding vertex of a bit position with value 1 in both W_3 and W_7 is a "thrown-away" vertex since a 1 in W_3 indicates that the vertex has been assigned a count of 1 and a 1 in W_7 indicates that the vertex has been assigned a count of 0. The contents of W_4 are now

$$W_4 \quad 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

11. Since no vertex has been thrown away in the step above, a new set of front vertices is established by copying W_0 to W_6. At the same time, the current count is replaced with the 2's complement value of the current count ($W_6 \leftarrow W_0$ and $W_1 \leftarrow\sim W_1$)). The contents of W_1 and W_6 are now

$$W_1 \quad 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

$$W_6 \quad 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0$$

12. The mark bits of number 1 in W_6 are encoded and the data read from the RAM are stored in W_0 ($W_0 \leftarrow$ RAM[MAR]). The contents of W_0 are now

$$W_0 \quad 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0$$

13. Assign the current count to each adjacent vertices. Since the current count stored in W_1 is 0, no logical operations are required to assign the current count. The contents of W_2 are now

$$W_2 \quad 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

14. Keep track of the vertices that have been assigned a count of 0 ($W_7 \leftarrow W_0 \wedge W_7$). The step is very similar to step 5. The contents of W_0 are shown below.

$$W_7 \quad 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0$$

15. Check for "thrown-away" vertices ($W_4 \leftarrow W_3 \wedge W_7$). The explanation is similar to that of step 10. The contents of W_4 are now

$$W_4 \quad 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0$$

16. Encode the mark bits of 1's in W_4 to clear the rows of the encoded addresses in the RAM. Since the "thrown-away" vertices are indicated by a 1 in the corresponding bit of W_4, the corresponding rows of the "thrown-away" vertices in the RAM can be cleared by encoding the mark bit of 1's in W_4.

17. At this step, the RAM is clear and the traversal is done. The counts of the vertices are stored in W_3. The corresponding vertices of the bits with 1's are assigned the first color, and the corresponding vertices of the bits with 0's are assigned the second color. The corresponding vertices of the bits with 1's in W_4 are thrown away. In this example, vertices A and C are assigned the first color, vertices B and E are assigned the second color, and vertex D is thrown away. The contents of W_3 and W_4 are now

$$W_3 \quad 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0$$

$$W_4 \quad 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0$$

In summary, the traversal of a graph consists of a number of repeated cycles. A cycle starts with encoding the front vertices for retrieving the data stored in the RAM, followed by assigning the current count to the adjacent vertices, accumulating the counts, and checking the thrown-away vertices. The operations are simple and can be done in a very small amount of time.

4.3.3 Organization of the Graph Traversal Unit

The graph traversal unit is a microprogrammed subsystem equipped with a local bus. The details of the graph traversal unit are shown in Figure 4.5. It consists of the following components:

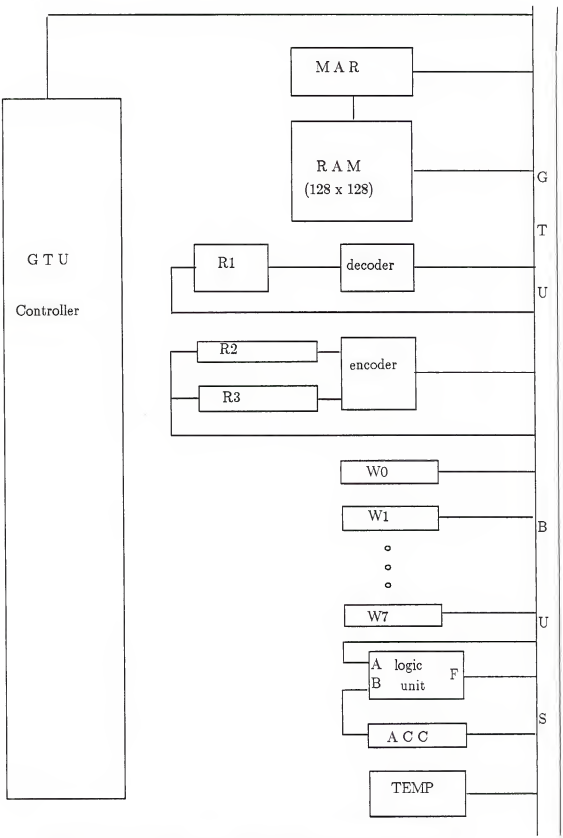


Figure 4.5. The Organization of the Graph Traversal Unit

1. Data memory and MAR: The data memory is a RAM of size 128 x 128 used to store the adjacency matrix of communication requests. MAR specifies the memory address for storing/retrieving data to/from the data memory.

2. Controller: It contains the micro-instructions for controlling the operations of a graph traversal.

3. Working registers (W0, W1,...,W7): The traversal of vertices involves exploring adjacent vertices, keeping track of front vertices, and checking for "thrown-away" vertices. The working registers are designed for marking and keeping track of the status of each vertex. A register is composed of 128 mark bits where each mark bit indicates the count or status of the corresponding vertex (communication request).

4. Logic unit and accumulator: This is where the logic operations are performed. The logic unit performs operations on two operands: one stored in the accumulator and the other stored in one of the working registers.

5. Priority encoder and temporary register: The priority encoder and address decoder are adopted from the work reported in [30,46]. Since the working registers that need to be encoded contain mostly 0's, it is very slow to check bit by bit for 1's. The priority encoder is used to encode the mark bits of registers and generate addresses for retrieving the data stored in the RAM. It works as follows. The priority encoder generates the address of the highest priority mark bit of 1. The generated address is loaded from the address bus to the RAM address register. The address is used to read from the RAM and to clear the mark bit. For example, the working register W_6 with the contents (W_6 0 1 0 1 1 0 0 0) indicates that vertices 1, 3, and 4 are the front

vertices. Priorities are assigned to these bits from left to right. Starting with the highest priority mark bit of 1 (bit 1), the address 00000001 is generated and the data stored at 00000001 of the RAM is retrieved. After the retrieval, the mark bit 1 is reset to allow the next highest priority bit to be encoded. The same procedure holds true for the mark bits 3 and 4. Another purpose of the priority encoder is to generate the address of the row in the RAM that needs to be cleared. For instance, for each of the "thrown-away" vertices, the address of the corresponding row in the RAM is generated by encoding the corresponding bit in W_4. Once the address is generated, the temporary register that is initialized to zero is used to clear the corresponding row in the RAM.

6. Address decoder: The main purpose of address decoder is to decode the address generated by priority encoder for resetting the mark bits. In the middle of the traversal, each of the front vertices has to be explored for adjacent vertices. After a front vertex has been explored, the address of that front vertex has to be decoded in order to reset the corresponding mark bit and allow the next mark bit of 1 to be encoded. In the example above, after the data stored in 00000001 has been retrieved, the address 00000001 has to be decoded by the address decoder to reset the mark bit and to allow the next mark bit of 1 to be encoded.

4.4 Performance Evaluation of the Partitionable Bus Networks

In this section, the performance of the dynamically partitionable bus network (DPBN) is compared with an "ideal" local area network using simulation. In addition, an aging analysis to avoid indefinite communication delay is discussed.

4.4.1 Simulation Study of DPBN

We will compare the performance of the dynamically partitionable bus network to an ideal network. An ideal local area network has a communication channel which has 100 percent utilization rate and is free from any request contention. In other words, it has the theoretically optimal performance of any one-communication-at-a-time bus network. The model of dynamically partitionable bus network used in the simulation is described below:

1. N stations are connected to a shared communication channel in a multipoint configuration. The probability that a station will issue a communication request at any instant of time is p.

2. The probability that a station will issue two communication requests in a very short period of time as compared with the time interval of communication is very small.

3. All communication requests take the same amount of time.

4. Communication requests are independent of each other.

In general, this model is characterized by two parameters: the number of stations connected to the bus (N) and the probability that a station will issue a communication request (p).

A simulation program that models the partitionable bus networks was implemented on a VAX 8600. The following steps describe the program:

1. For each station, if the number generated by the random number generator is less than p, a request is established. The destination of the request is determined by dividing the range [0 , 1) into N (the number of stations attached to

the bus) half open, equal intervals, with each interval representing a station, and the corresponding station of the interval in which the random number falls is the destination of the request.

2. An adjacency matrix is established by determining the conflicts among communication requests according to the procedure given in Section 4.1.

3. The graph traversal algorithm is applied to the adjacency matrix.

4. The number of colors assigned to vertices is the communication delay in units.

The example below shows how the communication requests are generated and processed by the program. A number of requests, as shown in Table 4.5, are established by a random number generator. The adjacency matrix shown in Table 4.6 is

Table 4.5. The Table of Communication Requests

Sending Station	Receiving Station
20	4
5	10
1	3
8	5
9	4
7	13
3	18
12	8
15	4
16	18

constructed according to the procedure in Section 4.3. A 1 in the cell indicates a conflict between the request on the row and the request on the column. For instance, the cell (4 , 1) is 1 because the request 4 (station 8 sends messages to station 5) is in conflict with request 1 (station 20 sends messages to station 4).

Table 4.6. The Adjacency Matrix of the Vertices in a Graph Corresponding to the Communication Requests in Table 4.5 (before graph traversal)

0	1	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	0
0	0	0	0	0	0	1	0	0	0
1	1	0	0	1	1	1	1	1	0
1	1	0	1	0	1	1	1	1	0
1	1	0	1	1	0	1	1	1	0
1	1	1	1	1	1	0	1	1	1
1	1	0	1	1	1	1	0	1	0
1	1	0	1	1	1	1	1	0	0
1	0	0	0	0	0	1	0	0	0

Table 4.7. The Discarded Vertices of the Graph Traversal on the Adjacency Matrix in Table 4.6

Discarded Vertices
4, 5, 6, 7, 8, 9

After the first graph traversal, the vertices, as shown in Table 4.7, are discarded.

A new adjacency matrix of the discarded vertices of the first graph traversal, as shown in Table 4.8, is established.

Table 4.8. The Adjacency Matrix of the Vertices in a Graph Corresponding to the Communication Requests in Table 4.7 (after the first graph traversal)

0	1	1	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	0

The vertices discarded in the second traversal and their adjacency matrix are shown in Tables 4.8 and 4.9, respectively.

Table 4.9. The Discarded Vertices of the Graph Traversal on the Adjacency Matrix in Table 4.8

Discarded Vertices
3, 4, 5, 6

Table 4.10. The Adjacency Matrix of the Vertices in a Graph Corresponding to the Communication Requests in Table 4.9 (after the second graph traversal)

0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0

In the third traversal, only vertex 4 is thrown away. The total number of colors used is seven and the communication delay is seven units.

Table 4.11 shows the expected communication delays and standard deviations in units of time frames for N = 25, 50, 75, 100 and p = 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9. A time frame is the duration of each communication request. The numbers in parentheses are the communication delays for an ideal network. The expected communication delays for the DPBN are obtained by taking the average of 20 simulation runs. The communication delay for each run is calculated by polling each station for a request, constructing a conflicting graph, and traversing the constructed graph. The expected communication delay for the ideal network is obtained by multiplying N by p. As shown in the table, the DPBN has much shorter communication delay.

Table 4.11. The Expected Communication Delays and Standard Deviations in Units for a DPBN and an Ideal Bus Network (in parentheses)

D_1/σ (D_2)	N = 25	N = 50	N = 75	N = 100
p = 0.2	3.10/1.18 (5.0)	6.50/2.85 (10.0)	9.30/2.49 (15.0)	11.40/2.93 (20.0)
p = 0.3	5.00/1.34 (7.5)	10.20/2.17 (15.0)	13.30/2.34 (22.5)	17.00/3.09 (30.0)
p = 0.4	6.80/1.77 (10.0)	12.90/2.60 (20.0)	17.40/3.08(30.0)	23.60/4.34 (40.0)
p = 0.5	8.80/2.04 (12.5)	14.75/2.19 (25.0)	22.80/2.89 (37.5)	29.20/4.31 (50.0)
p = 0.6	9.60/2.03 (15.0)	18.75/2.81 (30.0)	25.40/2.20 (45.0)	35.90/4.39 (60.0)
p = 0.7	11.20/1.82 (17.5)	19.20/3.73 (35.0)	30.20/3.43 (52.5)	38.80/2.94 (70.0)
p = 0.8	12.35/2.57 (20.0)	22.80/2.63 (40.0)	32.10/2.72 (60.0)	40.57/5.14 (80.0)
p = 0.9	13.60/1.41 (22.5)	26.40/3.63 (45.0)	37.80/2.92 (67.5)	51.00/5.24 (90.0)

In Table 4.11, D_1 and D_2 are the expected communication delays for a DPBN and an ideal network, respectively, σ is the standard deviation of the expected communication delays for a DPBN, N is the number of stations attached to the network, and p is the probability that a station may issue a communication request.

Table 4.12 gives the improvement ratio for parameters N and p. It was computed by the formula below:

$$\delta = \frac{D_2 - D_1}{D_1}$$

where δ= The improvement ratio

The improvement ratio the percentage of decrease of network delay of a DPBN as compared to an ideal network. It is obvious that with a little extra hardware, the DPBN reduces network delay by around 70 percent depending on the number of stations attached to the network and the frequency of requests.

Table 4.12. The Improvement Ratio of a DPBN over an Ideal Network

δ	N = 25	N = 50	N = 75	N = 100
p = 0.2	0.61	0.54	0.61	0.75
p = 0.3	0.50	0.47	0.69	0.76
p = 0.4	0.47	0.55	0.72	0.70
p = 0.5	0.42	0.69	0.65	0.71
p = 0.6	0.56	0.60	0.77	0.67
p = 0.7	0.56	0.83	0.74	0.80
p = 0.8	0.62	0.75	0.87	0.97
p = 0.9	0.65	0.70	0.78	0.76

The results in Tables 4.11 and 4.12 are obtained based on the assumption that new vertices (communication requests) are not allowed to join a graph traversal until all the old vertices (communication requests) have been assigned colors and processed. As each traversal continues, the size of the graph reduces and so does the number of communication processes allowed in the bus. It should be noted that there is no obvious functional relationship between the improvement ratio and p in Table 4.12. The future work will include the study of the relationship between the improvement ratio and the parameters of the network (N and p).

Table 4.13 shows the improvement ratio for the case when new vertices are allowed to join the "thrown-away" vertices after each traversal. The improvement ratio increases significantly in comparison with that shown in Table 4.12 because the size of the graph is not reduced after each traversal. In this simulation, the number of new communication requests stays the same during the traversals of a simulation run. In other words, the number of requests that are allowed to proceed in a traversal is the same as the number of newly added requests. When the number of communication requests allowed to go through reaches the number of initially generated requests, the simulation run stops. However, this strategy does not reduce the network delay of an ideal network, since the number of new requests does not affect the performance of an ideal network.

One problem resulting from this improvement is the starvation problem: It is possible that some communication requests may be withheld indefinitely as the new vertices join the "thrown-away" vertices causing the old vertices to be discarded repeatedly. In the next subsection, we shall propose a strategy to overcome this shortcoming.

Table 4.13. The Improvement Ratio of a DPBN over an Ideal Network When New Requests are Allowed to Join Traversals

δ	N = 25	N = 50
p = 0.2	0.43	0.70
p = 0.3	0.57	1.00
p = 0.4	0.71	1.55
p = 0.5	0.75	1.65
p = 0.6	1.20	1.90
p = 0.7	1.20	1.90
p = 0.8	1.25	2.20
p = 0.9	1.45	2.25

4.4.2 A Strategy for Solving the Aging Problem in DPBN

To allow the communication requests that have been held up for the longest period of time to have the highest priority in the next processing of communication requests, we propose the following strategy:

1. Maintain a priority queue that keeps track of the priorities of communication requests. Before any communication request starts transmitting messages, apply the graph traversal algorithm repeatedly to the thrown-away vertices and label the vertices with a time stamp. A time stamp is composed of the number of traversals and the assigned color. The priority queue is constructed in accordance with the time stamp. A typical priority queue looks like this: 1A, 1B, 2A, 2B, 3A..., where 1A is the group of vertices that are assigned the first color at the very first traversal and 1B is the group of vertices that are assigned the second color at the very first traversal, etc.

2. Each time the graph traversal algorithm is executed, a group of vertices, such as the vertices with priority 1A, are chosen to join the new communication requests and all the vertices with priority 1A are taken as the starting vertices of the graph traversal. Since none of the vertices with priority 1A are adjacent to each other and all are starting vertices, they are guaranteed to get a color in this graph traversal. At this point, the communication requests corresponding to the vertices in 1A and some new communication requests are allowed to go through. In the next traversal, the vertices with priority 1B will join the new requests and all the vertices in 1B become starting vertices. This procedure is continued until the priority queue is empty and a new queue is formed.

Figure 4.6 shows an example of this strategy. First, apply the graph traversal algorithm to the graph. Vertices A and C, colored with the first color, are in group

1A, vertices B and D, colored with the second color, are in group 1B, and vertex E, which was thrown away in the first traversal, is in group 2A. Second, combine a group of vertices from the priority queue in the order of priority with new communication requests for a graph traversal. This plan not only increases system throughput but also solves the problem of starvation.

4.5 Summary

In this chapter an application of the graph traversal algorithm to the dynamically partitionable bus network (DPBN) is presented. In a dynamically partitionable bus network a bus is partitioned into a number of subnetworks, and multiple communication processes are carried out at any moment. The proposed approach resolves conflicts by assigning colors to the graph where vertices represent communication requests and edges denote the conflicts among the communication requests. The communication requests corresponding to the vertices assigned the same color are carried out in parallel within subnetworks. In addition, a hardware design of graph traversal algorithm to speed up the identification and scheduling of non-conflicting communication requests is described. The performance evaluation of the partitionable bus network shows a significant increase of network throughput as compared with an ideal non-partitionable network in which no communication contention is assumed.

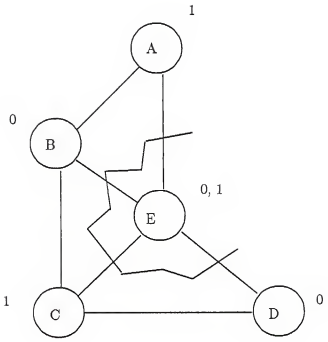


Figure 4.6. The Graph Traversal for the Strategy of Handling Aging

CHAPTER 5
DPBN AND COIN-CHANGING ALGORITHMS

One of the characteristics of a distributed system is that processors compete for resources and processes are performed in parallel. Due to the fact that the elapsed times of these processes vary, system resources are often idle as these processes are completed and resources are released at different times. For instance, in a partitionable bus network, requests that correspond to the vertices that have been assigned the same color proceed in parallel. When a communication is over, a subnetwork assigned to that communication process will be idle and has to wait for other parallel communication processes to complete before it can be reused in the next round of graph traversal and assignment. For instance, three communication requests are selected from a pool of communication requests $(q_1, q_2, q_3, ..., q_n)$ as shown in Table 5.1 by the graph traversal algorithm for parallel processing. The durations for q_1, q_3, and q_4 are 23, 189, and 288 time units, respectively. The process q_1 will finish first and be followed by q_3 and q_4. Since the requests do not finish at the same time, the partitioned subnetworks will not be fully utilized. The utilization rate for this example is

$$\frac{23 + 189 + 288}{288 * 3} = 0.578$$

The percentage of bus idling is approximately 42.

In a random distribution of the lengths of communication requests, the waste of bandwidth due to bus idling can reach 50 percent, since the expected length of communication requests is a half of the maximum length. The amount of idling

Table 5.1. An Example of Bus Idling

communication request	Sending Station	Receiving Station	duration of request
q_1	1	3	23
q_2	2	4	50
q_3	4	5	189
q_4	6	7	288
q_5	1	4	211
⋮	⋮	⋮	⋮
q_n	1	5	100

time can be reduced if the message of each request is divided into a number of small time frames. Each communication request is allowed to transmit messages for the duration of a time frame and the scheduling of message transfer is based on these time frames. There are two ways to divide a message into time frames. One way is to use a fixed-size time frames. Using this method, a message is divided into a number of fixed-size frames. For example, if each time frame is 23 units, q_1, q_3, and q_4 in this example can be scheduled for parallel processing with three subnetworks during the first time frame. q_3 and q_4 can be scheduled for parallel processing with two subnetworks during the next 8 time frames. q_4 can be completed in the next 4 time frames. Since subnetworks can be released as soon as communication requests are completed, the utilization of the communication channel can be increased. However, if a large frame size is used, some idle time would still remain. For example, in the ninth time frame in this example, the subnetwork assigned to q_3 would be idle for 18 time units since q_3 has only 5 time units in the last frame. If a small frame size is used, the overhead as a result of increased graph traversals and scheduling can be significant.

Another way to divide a message is to use time frames of variable sizes. Using this method, a message is partitioned into a number of frames of variable sizes. If the set of time frames of variable sizes is chosen properly it is possible to keep the number of time frames required for processing the communication requests to a minimum, thus reducing the number of graph traversals and process scheduling. If we select a number of time frames of different lengths and assign time frames to each individual communication process using the Greedy algorithm described in Chapter 2, where the total length of the assigned time frames equals the elapsed time of the communication process, the number of time frames assigned would be minimized and so is the number of graph traversals. The Greedy algorithm is simple and fast in comparison to other algorithms.

This problem of assigning the proper mix of time frames of variable sizes is the same as the coin-changing algorithm described in Section 2.4. The Greedy algorithm takes as many of the largest coin type (time frame) as possible and then as many of the second largest coin type as possible, etc. However, it has two disadvantages. First, the Greedy algorithm does not always generate an optimal solution for a given total time T. Second, in some cases, the Greedy algorithm does not generate a solution even though an optimal solution exists. The examples can be found in Section 2.4. Both disadvantages can be eliminated if the set of coin types (time frames) is chosen properly, for example, if the set of coin types $W = (W_1, W_2, W_3, ..., W_n)$ is in the form of a geometric sequence. A geometric sequence is defined as follows:

Definition 5.0.1 *Geometric Sequence: A sequence* $W = (W_1, W_2, W_3, ..., W_n)$, *where* $W_i < W_{i+1}$ *is geometric if* $W_{i+1}/W_i = r$ *and* r *is a positive integer.*

Since the problem of minimizing the number of coins is the same as the problem of minimizing time frames, the set of time frame sizes is chosen to be a geometric

sequence in our approach. There are two advantages to use a geometric sequence as the set of frame sizes. First, it guarantees that the Greedy algorithm always generates a solution if a solution exists, as proven in [7]. Second, as we shall show, the Greedy algorithm always produces an optimal solution if the set of frame sizes is a geometric sequence. The proof for the second point is given in the next section. However, for a given distribution of communication lengths, there exist many possible geometric sequences. For instance, in a normal distribution of lengths of communication requests with a maximum length 100 and a minimum length 1, some useful geometric sequences are as follows:

$W = (1, 2, 4, 8, 16, 32, 64)$

$W = (1, 3, 9, 27, 81)$

$W = (1, 4, 16, 64)$

The optimal set of frame sizes can be determined by simulation.

This chapter is organized as follows: Section 5.1 presents and proves some theorems related to the Greedy algorithm for coin-changing. Section 5.2 presents the application of coin-changing algorithm in partitionable bus networks. A performance evaluation is given in Section 5.3.

5.1 Some Theorems Related to Coin-Changing Algorithm

The terms used in this chapter are defined below. Some of these terms and definitions are taken from [7].

Definition 5.1.1 Representation: A set of non-negative integers $(X_1, X_2, ..., X_n)$ *which satisfies* $\sum_{i=1}^{n} X_i W_i = T$ *is called a representation of T with respect to the set of coin types* $(W_1, W_2, W_3, ..., W_n)$.

For example, X = (1, 2, 0, 1) is a representation of T = 17 with respect to W = (1, 3, 4, 10).

Definition 5.1.2 Canonical Representation: A representation obtained by taking the highest value coin type as many times as possible, then taking the next highest value coin type as many times as possible, etc. In other words, the canonical solution is the representation computed by the Greedy algorithm.

For example, the canonical representation for T = 17 in the example of Definition 5.1.1 is X = (0, 1, 1, 1).

Definition 5.1.3 Complete: A set of coin types is said to be complete if any given integer T has a canonical representation.

For example, the set W = (1, 3, 8, 10, 26) is complete, while W' = (2, 4, 8, 10) is not, since T = 13 has no representation with respect to W'.

Definition 5.1.4 Essentially Complete: A set of coin types $W = (W_1, W_2, W_3, ..., W_n)$ *is said to be essentially complete if whenever a constant T is representable, it always has a canonical representation.*

For example, W = (3, 9, 12, 27) is essentially complete, since any T that has a representation also has a canonical representation. W' = (3, 7, 12, 29) is not essentially complete since T = 9 has a representation (3, 0, 0, 0) with respect to W' but does not have a canonical representation.

Definition 5.1.5 Optimal Representation: A representation of an integer T with respect to $W = (W_1, W_2, W_3, ..., W_n)$ *is an optimal representation if it uses a minimal number of coins.*

Definition 5.1.6 Canonical Set: A set of coin types $W = (W_1, W_2, W_3, ..., W_n)$ *is canonical if each canonical representation of T with respect to W is optimal.*

Using the above definitions, we shall now present and prove a number of theorems.

The theorem below provides a sufficient condition to determine if a set of coin types is essentially complete.

Theorem 5.1.1 *[7] $W = (W_1, W_2, W_3, ..., W_n)$ is essentially complete if and only if $W_i = K_i W_1$, for $1 < i \leq n$, where K_i is a non-negative monotonically increasing integer.*

Proof of Theorem 5.1.1 *The proof is given in [7].*

Now, we present and provide a theorem that provides a sufficient condition for a cannonical set.

Theorem 5.1.2 *$W = (W_1, W_2, W_3, ..., W_n)$ is a canonical set of coin types if the sequence W_i $(i = 1, ..., n)$ is a geometric sequence with ratio $r \geq 1$ where $r = W_{k+1}/W_k$ and r is an integer.*

Proof of Theorem 5.1.2 *The strategy we take to prove the theorem is as follows. We want to show that if $X = (X_1, X_2, X_3, ..., X_n)$ is a representation of T generated by the Greedy algorithm with respect to the set of coin types $W = (W_1, W_2, W_3, ..., W_n)$ and $Y = (Y_1, Y_2, Y_3, ..., Y_n)$ is an optimal representation, then $X = Y$.*

Before we prove the theorem, we make the following four observations:

1. *$Y_i < r$ for $i = 1...n$. If Y_i is greater than r, we can simply replace r coins of type W_i with a single coin of type W_{i+1}. By doing so, we reduce the total number of coins used by r-1. (i.e., Y is not an optimal representation if any Y_i is greater than or equal to r.)*
2. *If $X_k \neq Y_k$ where k is the largest number less than n, then $X_k \geq Y_k$. This is because the Greedy algorithm always takes a maximum possible number of coins*

of the highest values. It is obvious that X_k is greater than Y_k at least by 1 if $X_k \neq Y_k$.

3. $\sum_{i=1}^{k} Y_i W_i < (Y_k + 1)W_k$.

   To show that the above inequality holds, we first replace Y_i by (r - 1) in $\sum_{i=1}^{k} Y_i W_i$ to obtain the following equation:

$$\sum_{i=1}^{k} Y_i W_i \leq \sum_{i=1}^{k-1} (r-1)W_i + Y_k W_k \qquad (1)$$

   The above equation holds because $Y_i < r$ according to observation 1.

   By taking the summation of the geometric series W_i, we obtain

$$\sum_{i=1}^{k-1} (r-1)W_i = W_k - 1 \qquad (2)$$

   .

   By substituting (2) into (1) we conclude

$$\sum_{i=1}^{k} Y_i W_i < (Y_k + 1)W_k$$

   .

4. $\sum_{i=1}^{n} X_i W_i = \sum_{i=1}^{n} Y_i W_i = T$. *The above equation is valid, since both X and Y are representations of T.*

Based on the above four observations, we now prove the theorem by contradiction.

We first assume that $X \neq Y$. If k is the largest number such that $X_k \neq Y_k$, the equation in observation 4 can be reduced to

$$\sum_{i=1}^{k} X_i W_i = \sum_{i=1}^{k} Y_i W_i$$

By observation 3, we have

$$\sum_{i=1}^{k} Y_i W_i < (Y_k + 1)W_k \qquad (3)$$

It is obvious that

$$\sum_{i=1}^{k} X_i W_i \geq X_k W_k$$

Since $X_k > Y_k$ *by at least 1, due to observation 2, we know*

$$X_k W_k \geq (Y_k + 1) W_k$$

Thus,

$$\sum_{i=1}^{k} X_i W_i \geq (Y_k + 1) W_k \qquad (4)$$

By (3) and (4)

$$\sum_{i=1}^{k} X_i W_i > \sum_{i=1}^{k} Y_i W_i$$

This is a contradiction to the fact that

$$\sum_{i=1}^{k} X_i W_i = \sum_{i=1}^{k} Y_i W_i$$

We conclude that either $X = Y$ *or* $X = (X_1, X_2, X_3, ..., X_n)$ *is the optimal representation of T.*

The following two corollaries show that a geometrical sequence is essentially complete, and is complete if $W_1 = 1$.

Corollary 5.1.1 *$W = (W_1, W_2, W_3, ..., W_n)$ is essentially complete if W_n is a geometric sequence.*

Proof of Corollary 5.1.1 *The proof is obvious from Theorem 5.1.1.*

Corollary 5.1.2 *$W = (W_1, W_2, W_3, ..., W_n)$ is complete if and only if $W_1 = 1$.*

Proof of Corollary 5.1.2 *The proof is obvious from the definition of Complete.*

Theorem 5.1.3 Given a set of coin types $W = (W_1, W_2, W_3, ..., W_n)$, *if* $W^k = (W_1, W_2, W_3, ..., W_k)$ *is canonical and the representation of* $T = pW_k$ *is optimal, then* $W^{k+1} = (W_1, W_2, W_3, ..., W_k, W_{k+1})$ *is canonical. The constant p is determined by taking the ceiling of* W_{k+1}/W_k *(i.e.,* $p = \lceil W_{k+1}/W_k \rceil$*).*

It should be noted that W^k represents the subset of W consisting of the first k elements of W.

Proof of Theorem 5.1.3 The proof is given in [58].

Theorem 5.1.3 is also called the one-point theorem. It provides a fast way to check if a set of coin types is canonical. A set of coin types $W = (W_1, W_2, ..., W_n)$ is a canonical set if the first (n-1) elements of W is a canonical set, and the representation of $\lceil W_n/W_{n-1} \rceil * W_n$ is optimal. The same rule can be applied to determine if the first (n-1) elements of W is canonical. For instance, to determine that $W = (1, 4, 15, 20)$ is not a canonical set, the following steps are required:

Step 1: W^1 = (1) is canonical. $W^2 = (1, 4)$ and p = $\lceil \frac{4}{1} \rceil = 4$.

Step 2: For T = $4 * W_1 = 4$, the representation (0, 1) obtained by the Greedy algorithm is an optimal representation. According to the theorem, $W^2 = (1, 4)$ is a canonical set.

Step 3: W^3 = (1, 4, 15) and p = $\lceil \frac{15}{4} \rceil = 4$.

Step 4: For T = $4 * W_2 = 4 * 4 = 16$, the representation (1, 0, 1) obtained by the Greedy algorithm is an optimal representation. According to the theorem, W^3 = (1, 4, 15) is a canonical set.

Step 5: W^4 = (1, 4, 15, 20) and p = $\lceil \frac{20}{15} \rceil = 2$.

Step 6: For T = $2 * W_3 = 2 * 15 = 30$, the representation (2, 2, 0, 1) obtained by the Greedy algorithm is not an optimal representation since the representation (0, 0,

2, 0) for T = 30 uses fewer coins. As a result, $W = (1, 4, 15, 20)$ is not a canonical set.

The above theorems serve the following purposes:

1. To establish a general rule for the construction of a canonical set (Theorem 5.1.2).

2. To determine if a set of coin types is complete or essentially complete (Corollary 5.1.1 and Corollary 5.1.2).

3. To provide an easy way to check if a set of coin types is canonical (Theorem 5.1.3).

In summary, a set of geometric sequence $W = (W_1, W_2, ..., W_n)$ with $W_1 = 1$ is a good choice for the set of time frames in our approach, since each communication request can be decomposed into time frames by the Greedy algorithm and the number of time frames assigned is minimal.

5.2 The Application of the Coin-Changing Algorithm in Partitionable Bus Networks

As stated before, a dynamically partitionable bus network allows multiple communication processes to be carried out at the same time. However, the bus capacity is still not fully utilized, since the message lengths of the communication requests are not the same in general. Assuming the message lengths of communication requests are uniformly distributed, the loss of throughput due to non-uniform termination of communication processes could be 50 percent or higher. Bus idling can be reduced if the scheduling of requests is based on some variable-sized time frames assigned to the communication requests. Only these requests with communication durations representable by the assigned frames can participate in the graph traversal. However, the number of time frames assigned to the communication requests needs to

be minimized to reduce the number of graph traversals. The problem of minimizing the number of time frames can be transformed into a coin-changing problem if we regard the size of a time frame as the face value of a coin type and the message length of a request as the total dollar amount that the individual coins should add up to. The proposed approach is to select a set of coin types (time frames) according to the theorems provided in the previous section and use the simple Greedy algorithm to decompose each message length. The steps to be taken in this approach are as follows:

1. Select a set of time frames of different sizes that form a geometric sequence. The purpose of choosing a geometric sequence is to guarantee that a communication request of any length can be divided into the selected time frames using the Greedy algorithm. Since the Greedy algorithm is very simple, it will add little overhead to the communication request.

2. Apply the Greedy algorithm to each communication request. Each message length will thus be represented by a minimal number of time frames.

3. Start with the largest time frame, designate it as the tag frame and collect the communication requests which have at least one unit of the tag frame.

4. Transform the collected requests into a graph according to the techniques described in Chapter 3.

5. Apply the graph traversal algorithm to the constructed graph and allow the requests that correspond to the vertices with the same color to be carried out simultaneously.

6. Repeat steps 3, 4, and 5 for the next largest time frame.

The example below demonstrates the approach:

Suppose there are m communication requests with lengths $l_1, l_2, l_3, ..., l_m$, respectively, as shown in Table 5.2. By applying the Greedy algorithm to each length of communication request with respect to the set of time frames $(m_1, m_2, m_3, ..., m_n)$, the time of each communication request is decomposed into time frames of different lengths. For instance, l_1 is made up of 3 frames of m_1, 1 frame of m_2, 2 frames of m_{n-1}, and 1 frames of m_n as shown in the Table 5.1. Next, all the requests having some units of time frame m_n are collected for graph traversal, and the requests assigned with the same color are allowed to send messages of length m_n. Continue this process for each time frame $m_{n-1}, m_{n-2}, ..., etc.$

The advantage of using the coin-changing algorithm with the graph traversal algorithm is obvious. The partitionable bus network can be more fully utilized. Since the Greedy algorithm always generates the optimal solution (i.e., the minimal number of time frames required to make up message length l_i with respect to m), the overhead of graph traversals can be reduced to a minimum.

Table 5.2. An Example of Decomposing Communication Requests

	l_1	l_2	l_3	l_4	$\cdots$	l_m
m_1	3	0	0	1	$\cdots$	2
m_2	0	2	1	1	$\cdots$	0
m_3	1	1	1	0	$\cdots$	4
m_4	0	5	2	2	$\cdots$	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m_{n-1}	2	0	3	0	$\cdots$	0
m_n	1	2	3	4	$\cdots$	1

l_i : The message length of comunication request i.
m_i : The size of the time frame i

5.3 Performance Evaluation

In this section, the usefulness of the proposed approach is demonstrated. The model and the parameters of the dynamically partitionable bus network are first described below:

1. The architecture of the partitionable bus network has been presented in Chapter 4. However, the lengths of communication requests are not uniform.

2. It is obvious that the distribution of the lengths of communication requests determines the effectiveness of using the coin-changing algorithm to maximize the utility of the dynamically partitionable bus network. In this performance evaluation, a normal distribution, an exponential distribution, and a uniform distribution are simulated.

   The appendices A and B show the probability distribution of a normal distribution and an exponential distribution, respectively.

3. The maximal and minimal lengths of the communication requests are 100 units and 1 unit, respectively.

4. The expected length of the communication requests is 50 units.

5. The time required to change switches between time frames is negligible.

In the simulation program, the set of frame sizes $(1, 2, 4, 8, 16, 32, 64)$ is used, which guarantees that the Greedy algorithm will generate the optimal solution. For the uniform distribution, the length of a communication request is determined by generating a random number between 0 and 1 and by multiplying that number by 100. Only one simulation run is completed for each distribution, and 100 communication

requests are generated for each run. For the normal distribution and the exponential distribution, the numbers are generated by using the procedure given in [16].

Table 5.3 shows the improvement ratio of a DPBN using time frames versus a DPBN without using time frames in a uniform distribution of communication lengths. The expected number of communication processes (ENP) being carried out at any instant is 2, 3, 4, 5, 6, 7, 8, 9, and 10, respectively. The improvement ratio Δ is the percentage of times the network delay is decreased. For instance, $\Delta = .30$ means that a decrease of network delay by 30 percent.

$$\Delta = \frac{T_1 - T_2}{T_1}$$

where T_1 is the total communication time of a DPBN without using time frames and, T_2 is the total communication time of a DPBN using time frames. As shown in Table 5.3, the network delay can be reduced by between 25 and 45 percent. The decrease of network delay is proportional to the ENP value.

Table 5.3. The Improvement Ratio of a DPBN Using Time Frames (Uniform Distribution of Communication Lengths)

ENP	T_1	T_2	Δ
2	6508	4798	0.2626
3	7381	4995	0.3232
4	8119	5076	0.3748
5	8295	5108	0.3841
6	8680	5053	0.4178
7	8836	5072	0.4260
8	8935	5081	0.4313
9	9138	5108	0.4409
10	9124	5086	0.4430

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames vs. a DPBN Without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Tables 5.4, 5.5, 5.6, and 5.7 show the improvement ratio in an exponential distribution of communication lengths. The exponent coefficient β equals 60, 40, 20, and 10, respectively.

Table 5.4. The Improvement Ratio of a DPBN Using Time Frames (Exponential Distribution of Communication Lengths with $\beta = 60$)

ENP	T_1	T_2	Δ
2	7163	5125	0.284
3	7752	4859	0.373
4	8277	4741	0.427
5	8741	4693	0.461
6	9134	4756	0.479
7	9325	4727	0.493
8	9475	4731	0.500
9	9573	4708	0.508
10	9720	4745	0.511

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.5. The Improvement Ratio of a DPBN Using Time Frames (Exponential Distribution of Communication Lengths with $\beta = 40$)

ENP	T_1	T_2	Δ
2	5575	3880	0.304
3	6119	3614	0.409
4	6728	3527	0.475
5	7261	3486	0.519
6	7715	3546	0.540
7	8066	3539	0.561
8	8299	3518	0.575
9	8404	3500	0.583
10	8722	3534	0.594

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.6. The Improvement Ratio of a DPBN Using Time Frames (Exponential Distribution of Communication Lengths with $\beta = 20$)

ENP	T_1	T_2	Δ
2	3060	2082	0.319
3	3381	1905	0.436
4	3837	1882	0.509
5	4166	1848	0.556
6	4553	1887	0.585
7	4868	1889	0.611
8	5061	1872	0.629
9	5232	1869	0.642
10	5493	1880	0.657

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.7. The Improvement Ratio of a DPBN Using Time Frames (Exponential Distribution of Communication Lengths with $\beta = 10$)

ENP	T_1	T_2	Δ
2	1577	1061	0.327
3	1735	960	0.446
4	1963	942	0.520
5	2129	920	0.567
6	2358	944	0.599
7	2517	944	0.625
8	2614	932	0.643
9	2698	930	0.655
10	2831	934	0.669

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Tables 5.8, 5.9, 5.10, 5.11, 5.12, and 5.13 show the improvement ratio for a normal distribution of communication lengths. The variance of the normal distribution varies between 5 and 40. The results show that, by partitioning a bus network to achieve parallel communication, the network throughput increases significantly when the variance of distribution is large. This is because when the variance is large, the lengths of the communication processes spread over a wider range resulting in a higher percentage of bus idling if the coin-changing algorithm is not used. When the variance is large, the application of the coin-changing algorithm in a dynamically partitionable bus network is justifiable.

Table 5.8. The Improvement Ratio of a DPBN Using Time Frames (Normal Distribution of Communication Lengths with $\sigma^2 = 40$)

ENP	T_1	T_2	Δ
2	6755	4959	0.265
3	7842	5156	0.342
4	8406	5192	0.382
5	8679	5091	0.413
6	9086	5055	0.443
7	9249	5066	0.452
8	9337	5041	0.460
9	9400	5078	0.461
10	9672	5066	0.476

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.9. The Improvement Ratio of a DPBN Using Time Frames (Normal Distribution of Communication Lengths with $\sigma^2 = 30$)

ENP	T_1	T_2	Δ
2	6741	5082	0.246
3	7391	4951	0.330
4	7871	5058	0.357
5	8132	5111	0.371
6	8373	5000	0.402
7	8553	4989	0.416
8	8739	4976	0.430
9	8875	5013	0.435
10	8967	4980	0.444

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.10. The Improvement Ratio of a DPBN Using Time Frames (Normal Distribution of Communication Lengths with $\sigma^2 = 20$)

ENP	T_1	T_2	Δ
2	6267	5133	0.180
3	6787	5109	0.240
4	7136	5082	0.287
5	7150	5017	0.298
6	7416	5021	0.323
7	7669	5023	0.345
8	7759	5010	0.354
9	7742	4957	0.259
10	7893	4963	0.371

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames
vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.11. The Improvement Ratio of a DPBN Using Time Frames (Normal Distribution of Communication Lengths with $\sigma^2 = 15$)

ENP	T_1	T_2	Δ
2	5851	5044	0.138
3	6168	4959	0.198
4	6502	4978	0.234
5	6671	5002	0.250
6	6868	5019	0.269
7	6690	5021	0.281
8	7073	5001	0.293
9	7098	4983	0.298
10	7186	4978	0.307

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames
vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.12. The Improvement Ratio of a DPBN Using Time Frames (Normal Distribution of Communication Lengths with $\sigma^2 = 10$)

ENP	T_1	T_2	Δ
2	5512	4940	0.103
3	5770	4962	0.140
4	5890	4965	0.157
5	6007	4964	0.173
6	6146	4970	0.191
7	6208	4963	0.200
8	6276	4954	0.210
9	6363	4962	0.221
10	6450	4964	0.230

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN with Time Frames
vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

Table 5.13. The Improvement Ratio of a DPBN Using Time Frames (Normal Distribution of Communication Lengths with $\sigma^2 = 5$)

ENP	T_1	T_2	Δ
2	5235	4973	0.050
3	5402	4969	0.080
4	5451	4974	0.087
5	5525	4966	0.101
6	5581	4960	0.119
7	5630	4974	0.116
8	5631	4960	0.119
9	5663	4954	0.125
10	5709	4958	0.131

T_1: The Total Communication Time of a DPBN without Using Time Frames
T_2: The Total Communication Time of a DPBN Using Time Frames
Δ: The Improvement Ratio of a DPBN Using Time Frames
vs. a DPBN without Using Time Frames
ENP: The Number of Communication Processes That Are Being Carried Out at Any Instant.
The Number of Requests Generated = 100
The Maximum Length of a Request = 100
The Minimum Length of a Request = 1
The Mean of the Requests = 50

In summary, the network delay that the DPBN reduces by using the coin-changing algorithm depends on the following two factors:

1. The expected number of communication processes (ENP). The ENP indicates the number of communication processes that are being carried out at any instant in the network. The higher the ENP, the better the chance of having large variation of communication lengths. As shown in all the tables above, Δ (the improvement ratio) increases with the increase of ENP values. The improvement ratio can be converted to the percentage of network delay decrease by using the following equation:

   percentage of network delay decrease $= \Delta * 100$

   $$= \frac{T_1 - T_2}{T_1} * 100$$

2. The variance (σ^2). As the variance of the distribution of communication lengths increases, the percentage of bus idling becomes higher in a DPBN without using time frames. This situation gives a DPBN that uses time frames a better performance since it minimizes bus idling.

It is also worth mentioning that the overhead of bus partitioning and switching may affect choice of frame length sequence as well as performance.

CHAPTER 6
SUMMARY, CONCLUSION, OTHER APPLICATIONS, AND FUTURE WORK

In a multiprocessor/multicomputer system, computational tasks are carried out in parallel. Limited system resources such as communication channels, memory devices, and processors are shared among concurrent processes. The performance of a computer system can decrease rapidly due to resource contentions. The problem of resource contention and allocation can be represented as a graph coloring problem, and the techniques for coloring graphs can be used to identify non-conflicting resource requests effectively.

6.1 Summary

This dissertation presents three graph traversal algorithms that give good estimations for the chromatic number of a graph, where the vertices of the graph represent events or requests and the edges connecting the vertices represent conflicts between events or requests. The complexity of each graph traversal algorithm is O (E), where E is the number of edges of the graph. The dynamic algorithm has better performance than the static algorithm, since it throws away vertices "on the fly" and results in few colors used. The analysis of the dynamic algorithm shows that it produces better results than the simulation results of the algorithm reported by Wood [61].

In this work, the dynamic graph traversal algorithm for detecting non-conflicting resource requests is applied in a dynamically partitionable bus network (DPBN) to partition the bus network into a number of subnetworks for processing sets of non-conflicting communication requests. It distinguishes conflicting and non-conflicting

requests by assigning different colors to vertices of a graph, which represent requests. The same color is assigned to a set of non-conflicting requests which can be scheduled for parallel processing in the subnetworks.

In this work, the design of a special-purpose processor (a hardware design of the dynamic algorithm) that can be used to speed up the scheduling of communication requests is also presented. The special processor receives from the control computer of a partitionable bus network the adjacency matrix, which indicates conflicts among requests. It applies the dynamic graph traversal algorithm and returns the identified non-conflicting requests to the control computer. The network is partitioned to allow non-conflicting requests to proceed in parallel. Performance evaluation of the partitionable bus network that uses the dynamic graph traversal algorithm is also carried out. The results show a significant decrease of network delay as compared with an ideal unpartitioned local area network in which no conflicts are assumed.

Since the communication processes that are allowed to proceed in the partitioned subnetworks take different time durations, the subnetworks that are released by the processes of shorter durations can not be reused until all the processes are completed. In other words, some subnetworks are not fully utilized. The performance of the dynamically partitionable bus network is further improved by reducing the subnetwork idling time. The subnetwork idling problem is solved by dividing the message of each request into a minimal number of variable-sized time frames using the coin-changing algorithm, and each communication request is allowed to transmit messages for the duration of a time frame. The coin-changing algorithm used is the Greedy algorithm which is simple and fast. It is shown in this work that, by choosing a proper set of coin types (time frames) such as a geometric sequence, the Greedy algorithm can always generate an optimal representation for a time requirement of a communication

request. The performance evaluation shows that the increase in network throughput depends on the the variance of the distribution of communication durations and the expected number of communication processes (ENP) that are being carried out at the same time interval. The higher the variance and the ENP, the greater the network throughput increase achieved by applying the coin-changing algorithm in a dynamically partitionable bus network.

6.2 Conclusion

This research makes the following four contributions. First, it introduces three new graph traversal algorithms and shows analytically that one of the algorithms can produce better results than the algorithm introduced by Wood. Second, it demonstrates that, by applying the dynamic graph traversal algorithm in a partitionable bus network, the throughput of the network can be dramatically increased. Third, it introduces several new theorems asserting that, if a set of coin types forms a geometric sequence, the Greedy coin-changing algorithm always generates an optimal solution. Fourth, the new algorithms, the analysis and simulation results, and the theorems introduced in this work have very general applications. They can be applied to solve any resource contention problem in a computer system and to improve its resource utilization and performance.

6.3 Other Applications

The approach presented in this dissertation has very broad and useful applications, since most real world problems that deal with resource sharing and scheduling can be interpreted as a graph coloring problem. For example, for database computers, such as MICRONET [51,53], MDBS [21], DBC/1012 [57], GAMMA [12],

HYPERTREE [10,47], REPT [49,50], NON-VON [20,48], Michigan's Boolean Cube-connected Multicomputer [2], GRACE [27,28,29], and DBMAC [38,39], the congestion of the communication bus due to the large amount of data transfer slows down the processing of database operations. The problem can be solved by partitioning the bus to increase network throughput and by minimizing unnecessary data transfer by identifying the sequence of operations which cause a large amount of data transfer. It works as follows. Apply the graph traversal algorithm to the graph, where vertices represent database operations and an edge is drawn between a pair of vertices if the corresponding database operations do not require a large amount of data transfer. Thus, the corresponding operations of a pair of vertices with an edge in between can be executed consecutively with minimal delay.

6.4 Future Work

This dissertation presents an alternative method for solving resource contention problems. It touches upon many areas. Some of the things we have done can be further refined and improved. They are as follows:

- Simulate the dynamic graph traversal algorithm and Wood's algorithm on the same set of randomly generated graphs for comparison.

- Investigate some open problems in graph coloring and coin-changing algorithms, such as three coloring, heuristic rules of coloring, and sufficient conditions of a coin set that results in an optimal representation when Greedy algorithm is applied.

- Compare the improvement of time frames of variable sizes and time frames of a fixed size on a DPBN. Decomposing the length of a request into time frames of variable sizes solves the idling problem. However, the overhead of running

the Greedy algorithm is a waste of bandwidth. An alternative is to decompose the length of a request into time frame of a fixed size. The overhead is reduced, but the number of graph traversals may be increased.

- Do more simulation runs on the simulation study of the DPBN in Chapters 4 and 5.
- Refine the model of DPBN in Chapter 5 to include bus switching time for performance evaluation.
- Study the strategy that allows non-conflicting requests with sizes of time frames smaller than that of the tag frame to join graph traversals.

APPENDIX A
PROBABILITY DISTRIBUTION: NORMAL DISTRIBUTION

A random variable X is said to have a normal distribution with mean μ and variance σ^2 if X has the probability distribution function given below:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

where $-\infty < x < \infty$ and e is a constant ($e \approx 2.7183$).

A procedure that generates a random distribution is in the next page.

The procedure [16] below generates a normal distribution defined in the previous page.

Procedure Normal Distribution;

(* *This Procedure Generates Numbers of Normal Distribution* *)

Begin

$A := 0$

$C := 2 * \pi$

If $(A = 0)$ *Then*

Begin

U :=U (0, 1); (*U *is the random number generator**)

V :=E(1); (*E *is the exponential number generator**)

$B := (2V)^{1/2}$

$U := CU$

$W_1 := BcosU$

$W_2 := BsinU$

$A := 1$;

Return W_1

End;

Else

Begin

$A := 0$

Return W_2

End;

End; (**Normal Distribution**);

APPENDIX B
PROBABILITY DISTRIBUTION: EXPONENTIAL DISTRIBUTION

A random variable X is said to have an exponential distribution if X has the probability distribution function given below:

$$f(x) = \begin{cases} (1/\beta)e^{-x/\beta} & \text{if } 0 \leq x < \infty \\ 0 & \text{otherwise} \end{cases}$$

A procedure that generates an exponential distribution is in the next page.

The procedure [16] below generates an exponential distribution defined in the previous page.

Procedure Exponential Distribution;
(* *This Procedure Generates Numbers of Exponential Distribution* *)
Begin
U :=U $(0,1)$(* U *is Random Number Generator* *)
$W := -ln(U)$
Return W
end; (* *Exponential Distribution* *)

REFERENCES

[1] C. K. Baru and S. Y. W. Su. Performance of Statistical Aggregation Operations in the SM3 system. Proceedings of ACM/SIGMOD International Conference on Management of Data, Association for Computing Machinery, Boston, June 1984.

[2] C. K. Baru and O. Frieder. Implementing Relational Database Operations in a Cube-connected Multicomputer. Proceedings of the International Conference on Data Engineering, IEEE, Los Angeles, Feb. 1986.

[3] C. K. Baru and S. Y. W. Su. The Architecture of SM3: A Dynamically Partitionable Multicomputer System. IEEE Transactions on Computers, Vol. C-35, No.9, Sept. 1986, pp. 790-801.

[4] B. Bollabos and A. G. Thomason. Random Graphs of Small Order. Annals of Discrete Math., Vol. 28, 1985, pp. 47-97.

[5] D. Brelaz. New Methods to Color the Vertices of a Graph. Communications of the ACM, Vol. 22, No. 4, 1979, pp. 251-256.

[6] R. L. Brooks. On Coloring the Nodes of a Network. Proceedings of Cambridge Philos. Soc., Vol. 37, 1941, pp. 194-197.

[7] L. Chang and J. F. Korsh. Canonical Coin Changing and Greedy Solutions. Journal of the Association for Computing Machinery, Vol. 23, No. 3, July 1976, pp. 418-422.

[8] S. K. Chang and A. Gill. Algorithmic Solution of the Change-Making Problem. Journal of the Association for Computing Machinery, Vol. 17, No. 1, Jan. 1970, pp. 113-122.

[9] N. Christofides. An Algorithm for the Chromatic Number of a Graph. Computer Journal, Vol. 14, No. 1, Feb. 1971, pp. 38-39.

[10] A. M. Despain and D. A. Patterson. X-tree: A Tree Structured Multiprocessor Computer Architecture. Proceedings of the Fifth Symposium on Computer Architecture, Palo Alto, California, 1978.

[11] D. DeWitt. A Multiprocessor Organization for Supporting Relational Database Management Systems. IEEE Transactions on Computers, Vol. C-28, No. 6, June 1979, pp. 395-406.

[12] D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna. GAMMA: a Performance Dataflow Database Machine. Proceedings of the 12th International Conference on Very Large Databases, Information Processing Society of Japan, Kyoto, Japan, Aug. 1986.

[13] T. Fei, C. K. Baru, and S. Y. W. Su. SM3: A Dynamically Partitionable Multicomputer System with Switchable Main Memory Modules. Proceedings of the International Conference on Computer Data Engineering, IEEE, Los Angeles, April 1984.

[14] M. R. Garey and D. S. Johnson. The Complexity of Near-optimal Graph Coloring. Journal of Association for Computing Machinery, Vol. 23, 1976, pp. 43-49.

[15] M. R. Garey and D. S. Johnson. Computers and Intractability. W. H. Freeman and Company, San Francisco, 1979.

[16] G. S. Fishman. Principle of Discrete Event Simulation. John Wiley and Sons, New York, 1979.

[17] E. Hafner, Z. Nenadal, and M. Tschanz. A Digital Loop Communication System. IEEE Transactions on Communications, Vol. Com-22, No. 6, June 1974, pp. 877-881.

[18] S. L. Hakimi and Schineichel. Chromatic Factorizations of a Graph. Journal of Graph Theory, Vol. 12, No. 2, 1988, pp.177-182.

[19] J. L. Hammond and P. O'Reilly. Performance Analysis of Local Computer Networks. Addision Wesley Publishing Company, Reading, Mass., 1986.

[20] B. K. Hillyer, D. E. Shaw, and A. Nigram. NON-VON's Performance on Certain Database Benchmarks. IEEE Transactions on Software Engineering, Vol. SE-12, No. 4, April 1986, pp. 577-583.

[21] D. K. Hsiao and M. J. Menon. Design and Analysis of a Multi-backend Database System for Performance Improvement, Functionality Expansion, and Capacity Growth. Technical Report NPS52-83-006, Naval Postgraduate School, Monterey, California, June 1983.

[22] IEEE Project 802 Committee. A Status Report on Local Network Standards Committee. Draft C, May 1982.

[23] IEEE Standard 802.4. Token Passing Bus Access Method and Physical Layer Specifications. Draft D, 1982.

[24] A. Johri and D. W. Matula. Probabilistic Bounds and Heuristic Algorithms for Coloring Large Random Graphs. Technical Report, Department of Computer Science and Engineering, Southern Methodist University, Dallas, Texas, 1982.

[25] R. M. Karp. Reducibility among Combinatorial Problems. In R. E. Miller and J. W. Thatcher(eds), Complexity of Computer Computations, Plenum Press, New York, 1972.

[26] S. Kartashev and S. Kartashev. Dynamic Architecture: Problems and Solutions. Computer, Vol. 11, No. 7, July 1978, pp. 7-15.

[27] M. Kitsuregawa, H. Tanaka, and T. Moto-oka. Relational Algebra Machine GRACE. RIMS Symposia on Software Science and Engineering, 1982. Lecture Notes in Computer Science, Springer-Verlag, New York, 1983, pp. 191-212.

[28] M. Kitsuregawa, H. Tanaka, and T. Moto-oka. Architecture and Performance of Relational Algebra Machine GRACE. Proceedings of the International Conference on Parallel Processing, IEEE, Bellaire, Michigan, 1984, pp. 241-250.

[29] M. Kitsuregawa, M. Fushimi, H. Tanaka, and T. Motooka. In D. J. Dewitt and H. Boral,(eds.), Memory Management Algorithms in Pipeline Merge Sorter. Proceedings of the Fourth International Workshop on Database Machines, Springer-Verlag, New York, 1985, pp. 208-232.

[30] H. Lam, S. Y. W. Su, F. L. C. Seeger, C. Lee, and W. R. Eisenstadt. A Special Function Unit for Database Operations Within Data-Control Flow System. Proceedings of the International Conference on Parallel Processing, IEEE, St. Charles, IL., Aug. 1987.

[31] E. L. Lawler. A Note on the Complexity of the Chromatic Number Problem. Information Processing Letter, Vol. 5, 1976, pp. 66-67.

[32] F. T. Leighton. A Graph Coloring Algorithm for Large Scheduling Problem. Journal Res. Natn. Bur. Stand., Vol. 84, 1979, pp. 489-496.

[33] M. T. Liu. Design of the Distributed Double-loop Computer Network (DDLCN). Journal of Digital Systems, Vol. 5, 1981, pp. 3-37.

[34] M. Magazine, G. M. Nemhauser, and L. E. Trotter. When the Greedy Solution Solves a Class of Knapsack Problems. Operations Research, Vol. 23, 1975, pp. 207-217.

[35] D. Matula. On the Complete Subgraphs of a Random Graph. Proceedings of the Second Chapel Hill Conference on Combinatorial Mathematics and Its Application, University of North Carolina, Chapel Hill, 1970, pp. 356-369.

[36] D. Matula, G. Marble, and J. Isaacson. Graph Coloring Algorithms in Graph Theory and Computing. Academic Press, New York, 1972, pp. 109-122.

[37] R. M. Metcalfe, D. R. Boggs, C. P. Thacker, and B. W. Lampson. Multipoint Data Communication System with Collision Detection. U.S. Patent 4063220, 1977.

[38] M. Missikoff. A Domain-Based Internal Schema for Relational Database Machines. 1982 ACM-SIGMOD Conference, Santa Monica, California, 1982.

[39] M. Missikoff and M. Terranova. The Architecture of a Relational Database Computer Known as DBMAC. In David K. Hsiao, (ed.), Advanced Database Machine Architecture, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[40] J. Mitchem. On Various Algorithms for Estimating the Chromatic Number of a Graph. Computer Journal, Vol. 10, 1976, pp. 182-183.

[41] J. L. Mott, A. Kandel, and T. P. Baker. Discrete Mathematics for Computer Scientists. Reston Publishing Company, A Prentice-Hall Company, Reston, Virginia, 1983.

[42] R. M. Needham and A. J. Herbert. The Cambridge Distributed Computing Systems. Addision Wesley Publishing Company, Reading, Mass., 1982.

[43] R. Newman-Wolfe. Communication Issues in Parallel Computation. Ph.D. Dissertation, Computer Science Department, University of Rochester, Rochester, New York, 1986.

[44] J. R. Pierce. Network or Block Switching of Data. Bell System Technical Journal, Vol. 51, No. 6, pp. 1133-1145, 1972.

[45] J. Plesnik. Coloring of Graphs by Partitioning. Math. Slovaca, Vol. 30, No. 2, 1980, pp. 121-126.

[46] L. Raschid, T. Fei, H. Lam, and S. Y. W. Su. A Special Function Unit for Sorting and Sort-based Database Operations. IEEE Transactions on Computers (correspondence), C-35, No.12, Dec. 1986, pp. 1071-1077.

[47] C. H. Sequin, A. M. Despain, and D. A. Patterson. Communication in X-Tree, A Modular Multiprocessor System. Proceedings of the ACM, ACM, Austin, Texas, 1978.

[48] D. E. Shaw. A Parallel Algorithm for External Sorting, NON-VON Supercomputer. Technical Report, Computer Science Department, Columbia University, New York, August 1982.

[49] R. K. Shultz. A Multiprocessor Computer Architecture for Database Support. Ph.D. Dissertation, Department of Computer Science, Iowa State University, Ames, Iowa, 1981.

[50] R. K. Shultz and R. J. Zingg. Response Time Analysis of Multiprocessor Computers for Database Support. ACM Transactions on Database Systems, Vol. 9, No. 1, 1984, pp. 100-132.

[51] S. Y. W. Su, S. Lupkiewicz, D. H. Lo, and K. Doty. MICRONET: A Microcomputer Network System for Managing Distributed Relational Databases. Proceedings of the 4th International Conference on VLDB, Berlin, West Germany, 1978.

[52] S. Y. W. Su and K. P. Mikkilineni. Parallel Algorithms and Their Implementation in MICRONET. Proceedings of the 8th International Conference on VLDB, Mexico City, Sept. 1982.

[53] S. Y. W. Su. A Microcomputer Network System for Distributed Relational Databases: Design, Implementation, and Analysis. Journal of Telecommunication Networks, Vol. 2, No. 3, 1983, pp. 307-333.

[54] S. Y. W. Su and C. K. Baru. Dynamically Partitionable Multicomputers with Switchable Memory. Journal of Parallel and Distributed Computing, Vol. 1, Nov. 1984, pp. 152-184.

[55] S. Y. W. Su. Database Computers: Principles, Architecture, and Techniques. McGraw-Hill, New York, 1988.

[56] Y. Tanaka, Y. Noxaka, and A. Masuyama. Pipeline Searching and Sorting Models as Components of a Data Flow database Computer. Information Processing 80, North-Holland Publishing Co., Amsterdam, 1980.

[57] Terradata Corporation. DBC/1012 Data Base Computer Concepts and Facilities. Release 1.1, C02-0001-01, Los Angeles, 1984.

[58] B. N. Tien and T. C. Hu. Error Bounds and the Applicability of the Greedy Solution to the Coin-Changing Problem. Operation Research, Vol. 25, No. 3, 1977, pp. 404-418.

[59] C. C. Wang. An Algorithm for the Chromatic Number of a Graph. Journal of the Association for Computing Machinery, Vol. 21, No. 3, July 1974, pp. 385-391.

[60] D. J. A. Welsh and M. B. Powell. An Upper Bound to the Chromatic Number of a Graph and its Application to Time-Tabling Problem. Computer Journal, Vol. 10, 1967, pp. 85-86.

[61] D. C. Wood. A Technique for Coloring a Graph Applicable to Large Scale Time-Tabling Problem. Computer Journal, Vol. 12, 1969, pp. 317-319.

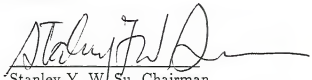
[62] J. W. Wright. The Change-Making Problem. Journal of the of the Association for Computing Machinery, Vol. 22, No. 1, Jan. 1975, pp. 125-128.

BIOGRAPHICAL SKETCH

Mr. Woo received the degrees of B.S. in engineering from National Taiwan University, M.S. in mechanical and aerospace engineering from the State University of New York at Buffalo, and M.S. in applied mathematics and computer science from Northwestern University in 1980, 1982, and 1985, respectively.

While at Northwestern University, Mr. Woo wrote two published papers in the area of mathematical modeling and simulation. During the period of pursuing the Ph.D. degree at the University of Florida, he has written two papers in the area of computer networks. After graduation he plans to concentrate his research on database management and computer networking.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

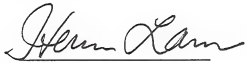


Stanley Y. W. Su, Chairman
Professor of Computer and Information Sciences and Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

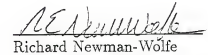
Yuan-Chieh Chow
Professor of Computer and Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Herman Lam
Associate Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Richard Newman-Wolfe
Assistant Professor of Computer and Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Chung-Yee Lee

Chung-Yee Lee
Assistant Professor of Industrial
and Systems Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

May 1989

Herbert A. Bruns

Dean, College of Engineering

Dean, Graduate School